ΑΝΑΠΤΥΞΗ
ΕΦΑΡΜΟΓΩΝ
ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΟ
ΠΕΡΙΒΑΛΛΟΝ

ΚΕΦΑΛΑΙΟ 1ο

ΑΝΑΛΥΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

ΠΕΡΙΕΧΟΜΕΝΑ

- **Η έννοια πρόβλημα**
- **Κατανόηση προβλήματος**
- **Δομή προβλήματος**
- **Καθορισμός απαιτήσεων**
- **Κατηγορίες προβλημάτων**
- **Πρόβλημα και υπολογιστής**

Εισαγωγή

Το πρόβλημα αποτελεί έννοια που απαντάται σε όλες τις επιστήμες και τους κλάδους τους, αλλά παράλληλα και στην καθημερινή μας ζωή. Τόσο η αντιμετώπιση, όσο και η διατύπωση ενός προβλήματος, αποτελούν διαδικασίες που απαιτούν ιδιαίτερες αναλυτικές και συνθετικές ικανότητες, ορθολογική σκέψη, αλλά και σωστό και εμπεριστατωμένο χειρισμό της φυσικής γλώσσας. Οι δεξιότητες που αποκτούνται από τους μαθητές μέσω της ενασχόλησής τους με την ανάλυση και τον ορισμό προβλημάτων, αποτελούν εφόδια γενικής χρηστικότητας, αφού μπορούν να λογίζονται σαν γνωστικά εργαλεία χρήσιμα για κάθε δραστηριότητα που είτε διαπερνά όλο το φάσμα των επιστημών, είτε αναφέρεται σε καθημερινές καταστάσεις.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι :

- να μπορούν καταρχήν οι μαθητές να κατανοούν πλήρως τα προβλήματα που τους τίθενται
- να μπορούν στη συνέχεια να προσδιορίζουν τα συστατικά μέρη ενός προβλήματος
- να μπορούν να αναλύουν ένα πρόβλημα σε άλλα απλούστερα
- να καταστούν ικανοί να προσδιορίζουν τα δεδομένα που τους παρέχονται για την αντιμετώπιση του προβλήματος
- να μπορούν να προσδιορίζουν τα ζητούμενα αποτελέσματα και τη μορφή απόδοσής τους
- να είναι σε θέση να θέσουν οι ίδιοι προβλήματα διατυπώνοντάς τα με ακρίβεια και πληρότητα

Προερωτήσεις

- ✓ θεωρείς σημαντικό το γεγονός να μιλάς και να γράφεις πολύ καλά τη φυσική γλώσσα στην προσπάθειά σου να επιλύσεις ένα τυχαίο πρόβλημα;
- ✓ έχεις ακούσει για "το πρόβλημα του έτους 2000";
- ✓ υπάρχει νοηματική διαφορά ανάμεσα στους όρους δεδομένο και πληροφορία;
- ✓ όταν αναφερόμαστε σε προβλήματα, έμμεσα δηλώνουμε την ανάγκη χρήσης υπολογιστών για την αντιμετώπισή τους;

1.1 Η έννοια πρόβλημα

Η καθημερινή εμπειρία και πρακτική μας εμφανίζει πολλά και ποικίλα προβλήματα που μας απασχολούν είτε στον προσωπικό μας χώρο, είτε στον κοινωνικό μας χώρο γενικότερα. Οι στατιστικές και οι δημοσκοπήσεις, που βλέπουν κατά καιρούς το φως της δημοσιότητας, καταγράφουν τα σημαντικότερα προβλήματα που απασχολούν το κοινωνικό σύνολο. Οι περισσότεροι από εμάς σήμερα, σε μια πιθανή έρευνα σχετική με τα κυριότερα προβλήματα που απασχολούν την ελληνική κοινωνία, θα απαντούσαμε πως ανάμεσα στα βασικότερα προβλήματα βρίσκονται η ανεργία, τα ναρκωτικά, η ξενοφοβία. Όπως επίσης σε μια παρόμοια έρευνα που θα επιχειρούσε να εντοπίσει τα κυριότερα παγκόσμια προβλήματα που απασχολούν την ελληνική κοινωνία, θα απαντούσαμε πως σαν κύρια προβλήματα θεωρούμε τον πόλεμο και τη μόλυνση του περιβάλλοντος.

Ορισμός : Με τον όρο **Πρόβλημα** εννοείται μια κατάσταση η οποία χρήζει αντιμετώπισης, απαιτεί λύση, η δε λύση της δεν είναι γνωστή, ούτε προφανής.

Άλλου είδους απαντήσεις, που θα βρισκόντουσαν σε άλλο φάσμα, άλλο εύρος, θα δίναμε αν η ερώτηση αφορούσε στα προσωπικά μας προβλήματα. Πολλοί από μας πιθανό να ανέφεραν επαγγελματικά προβλήματα με τον εργοδότη τους, προσωπικά προβλήματα στη σχέση τους ή προβλήματα που σχετίζονται με απλά θέματα της καθημερινής ζωής.

Η ύπαρξη προβλημάτων δεν αποτελεί χαρακτηριστικό γνώρισμα της εποχής μας. Μια απλή περιήγηση ανά τις σελίδες της ιστορίας, αρκεί για να μας επιβεβαιώσει ότι σε κάθε εποχή αναφέρονται προβλήματα διαφορετικής υφής και εμβέλειας.

- Ο Όμηρος στην Ιλιάδα περιγράφει το πρόβλημα που αντιμετώπιζαν οι Έλληνες πολιορκητές της Τροίας, μέχρι ο Οδυσσέας να επινοήσει το Δούρειο Ίππο.
- Το πρόβλημα μέτρησης του χρόνου, που αποτελούσε ταυτόχρονα ανθρώπινη ανάγκη, ήρθε να αντιμετωπίσει η εμφάνιση της κλεψύδρας και του εκκρεμούς.
- Προβλήματα κοινωνικής αδικίας και εκμετάλλευσης ήταν αυτά που οδήγησαν στην εμφάνιση του Robin Wood στα δάση της επαρχίας του Nottingham.
- Το πρόβλημα με το ψύχος που αντιμετώπισαν τα στρατεύματα του Ναπολέοντα στην εκστρατεία του στη Ρωσία, είχε σαν αποτέλεσμα την ανακοπή της προέλασης και την οπισθοχώρησή του.

Ένα από τα σημαντικότερα προβλήματα στο χώρο των υπολογιστών είναι αυτό που αναφέρεται σαν **πρόβλημα του έτους 2000 (millennium bug).** Το πρόβλημα εντοπίζεται στο ότι οι υπολογιστές μετρούν την ημερομηνία μόνο με δύο στοιχεία για κάθε ένα από τα τρία συνθετικά της. Για παράδειγμα, η ημερομηνία 15 Απριλίου 1999 συμβολίζεται με τον κωδικό 150499.

Από τις πρώτες περιόδους λειτουργίας των υπολογιστών, τότε που γινόταν κάθε δυνατή προσπάθεια να εξοικονομηθεί πολύτιμος αποθηκευτικός χώρος, καθιερώθηκε η καταγραφή της ημερομηνίας με τον παραπάνω τρόπο. Οπότε η πρώτη μέρα του 21ου αιώνα θα συμβολίζεται με τον κωδικό 010100, πράγμα που θα επιφέρει μεγάλη αναστάτωση και σύγχυση στους υπολογισμούς που θα πραγματοποιούν οι υπολογιστές.

Η αυγή του 2000 απειλεί να "τρελάνει" τους υπολογιστές. Οι αυτόματες μηχανές συναλλαγών (ATM) των τραπεζών μπορεί να μην δίνουν λεφτά ή να δίνουν απίστευτα ποσά που να μην ανταποκρίνονται στις πραγματικές καταθέσεις των πελατών. Τα μηχανογραφημένα λογιστήρια των επιχειρήσεων μπορεί να αποδίδουν τρελούς πίνακες οικονομικών στοιχείων που καμία σχέση να μην έχουν με την πραγματικότητα. Ένα ευρύ φάσμα κοινωνικών υπηρεσιών – ασφάλιση, υγειονομική περίθαλψη, παροχή ενέργειας, μεταφορές, κλπ - παρέχονται μέσα από χρήση πολύπλοκων υπολογιστικών συστημάτων, που απειλούνται από το πρόβλημα του έτους 2000.

Το ζήτημα όσο και αν φαίνεται απλό, στην πραγματικότητα είναι πολύπλοκο και κυρίως μπορεί να έχει πλευρές που δεν μπορούν να προβλεφθούν. Οι επιπτώσεις του προβλήματος του έτους 2000 μπορεί να είναι πολύ μεγάλες. Το πρόβλημα πρέπει να αντιμετωπιστεί με καθαρά τεχνικούς τρόπους, δεν παύει όμως να έχει και λειτουργικές, οικονομικές και νομικές προεκτάσεις.

- Σοβαρότατα προβλήματα επιδημιών, όπως η πανούκλα, η χολέρα και η λύσσα, αφάνιζαν καθημερινά χιλιάδες ανθρώπους τον περασμένο αιώνα μέχρις ότου επιστήμονες, όπως ο Pasteur και ο Fleming, να ανακαλύψουν τα κατάλληλα εμβόλια.
- Το πρόβλημα της μεταφοράς της ηλεκτρικής ενέργειας από τον τόπο παραγωγής στα σημεία κατανάλωσης πονοκεφάλιασε πολύ τους υπεύθυνους περασμένων εποχών μέχρι να εμφανιστούν οι μετασχηματιστές οι οποίοι έδωσαν λύση στο πρόβλημα.
- Το ενεργειακό πρόβλημα από την άποψη των αποθεμάτων που απα-

σχόλησε έντονα, αλλά και εξακολουθεί να απασχολεί την παγκόσμια κοινότητα, οδήγησε στην υιοθέτηση ήπιων μορφών ενέργειας, όπως είναι η ηλιακή ενέργεια, η αιολική ενέργεια και η βιομάζα.

- Το πρόβλημα της τρύπας του όζοντος, και κατ' επέκταση το πρόβλημα της προστασίας του φυσικού περιβάλλοντος, αντιμετωπίστηκε σε πρώτο βαθμό με τον περιορισμό εκπομπής χλωροφθορανθράκων από τις βιομηχανικές μονάδες που αποτελούν την κύρια αιτία του προβλήματος.
- Φυσικά φαινόμενα, όπως εκρήξεις ηφαιστείων, παλιρροιακά κύματα, σεισμοί και τυφώνες, αποτελούν σημαντικά προβλήματα ακόμα και στην εποχή μας, με αποτέλεσμα οι πληθυσμοί των περιοχών που πλήττονται να μετρούν ανθρώπινα θύματα, να υπόκεινται οικονομική καταστροφή και να αναγκάζονται πολλές φορές σε μετακίνηση.
- Ο υποσιτισμός ενός πολύ μεγάλου μέρους του πληθυσμού της αφρικανικής κύρια ηπείρου, οι καθημερινοί θάνατοι πολλών ανθρώπων, ειδικά μικρών παιδιών, αποτελεί ένα από τα σοβαρότερα προβλήματα της ανθρωπότητας σήμερα, χωρίς να έχει μπορέσει να αντιμετωπιστεί επαρκώς από τις ανθρωπιστικές οργανώσεις και τους διεθνείς οργανισμούς.
- Η αργή ταχύτητα μετάδοσης των δεδομένων σε σχέση με τις απαιτήσεις της σύγχρονης τεχνολογίας, αποτελεί ένα πρόβλημα που αντιμετωπίζεται σε ικανοποιητικό βαθμό από τη τεχνολογία των οπτικών ινών.
- Η ενοποίηση των τεσσάρων πεδίων δυνάμεων, του βαρυτικού, του ηλεκτρομαγνητικού, του ασθενούς πυρηνικού και του ισχυρού πυρηνικού, αποτελεί ένα πρόβλημα της σύγχρονης φυσικής που, προς το παρόν, δεν έχει επιλυθεί.

1.2 Κατανόηση προβλήματος

Η οποιαδήποτε προσπάθεια αντιμετώπισης ενός προβλήματος είναι καταδικασμένη σε αποτυχία αν προηγουμένως δεν έχει γίνει απόλυτα κατανοητό το πρόβλημα που τίθεται. Η κατανόηση ενός προβλήματος αποτελεί συνάρτηση δύο παραγόντων, της σωστής διατύπωσης εκ μέρους του δημιουργού του και της αντίστοιχα σωστής ερμηνείας από τη μεριά εκείνου που καλείται να το αντιμετωπίσει.

Η μορφή με την οποία παρουσιάζεται ένα πρόβλημα μπορεί να είναι ο-

ποιαδήποτε αρκεί να μπορεί να γίνει αντιληπτή από μία από τις πέντε ανθρώπινες αισθήσεις. Το πρόβλημα της ρύπανσης της ατμόσφαιρας της πρωτεύουσας μπορεί να το αντιληφθεί ο καθένας κοιτάζοντας τον αττικό ουρανό ή αναπνέοντας με δυσκολία ανηφορίζοντας κάποιο καλοκαιρινό μεσημέρι την οδό Ακαδημίας. Τα προβλήματα και τα δεινά που ταλαιπωρούν και σκοτώνουν χιλιάδες συνανθρώπους μας εξ αιτίας των πολεμικών συγκρούσεων στα διάφορα μέρη του κόσμου, μας γίνονται γνωστά είτε διαβάζοντας εφημερίδες, είτε ακούγοντας το ραδιόφωνο, είτε βλέποντας τηλεόραση.

Τα προβλήματα που μπορεί να κληθούμε να αντιμετωπίσουμε κατά τη διάρκεια της ζωής μας μπορούν να αναφέρονται σε οποιοδήποτε τομέα, μπορεί να αφορούν στα μαθηματικά, στη φυσική, στη λογική, στην καθημερινή ζωή ή οτιδήποτε άλλο θα μπορούσε κάποιος να σκεφτεί. Μπορεί να απαιτούνται γνώσεις συγκεκριμένων επιστημών ή μπορεί οι βιωματικές μας καταστάσεις και εμπειρίες να επαρκούν για την αντιμετώπισή τους. Μπορεί να είναι πολύπλοκα ή σχετικά απλά, πρωτόγνωρα ή συνηθισμένα. Σε κάθε όμως περίπτωση θα πρέπει να γίνουν απολύτως κατανοητά πριν γίνει κάθε προσπάθεια αντιμετώπισής τους.

Η κατανόηση ενός προβλήματος εξαρτάται σε μεγάλο βαθμό από την διατύπωσή του. Οποιοδήποτε μέσο μπορεί να χρησιμοποιηθεί για να αποδοθεί η διατύπωση ενός προβλήματος. Συνηθέστερο από όλα είναι ο λόγος, είτε ο προφορικός, είτε ο γραπτός.

Σαφήνεια διατύπωσης

Ο λόγος σαν μέσο επικοινωνίας και συνεννόησης πρέπει να χαρακτηρίζεται από σαφήνεια. Άστοχη χρήση ορολογίας, λανθασμένη σύνταξη, είναι δύο στοιχεία που μπορούν να προκαλέσουν παρερμηνείες και παραπλανήσεις. Η ικανότητα εκφοράς σωστού προφορικού και γραπτού λόγου αποτελεί μεγάλο προτέρημα για κάθε άτομο. Η παρερμηνεία είναι δυνατή ακόμα και σε περιπτώσεις όπου όλοι οι λεξικολογικοί και συντακτικοί κανόνες κρατούνται με ευλάβεια.

Παράδειγμα 1

Ένας πολυάσχολος επιχειρηματίας απευθύνεται στη σύζυγό του και της ζητά να φροντίσει για την αγορά αναμνηστικών δώρων για μερικούς παιδικούς του φίλους, που πρόκειται να συναντήσει μετά από πάρα πολλά χρόνια. Η σύζυγος του ζητάει να της δώσει κάποια χαρακτηριστικά γνωρίσματα των φίλων του, έτσι ώστε να γίνει πιο εύκολη η επιλογή των δώρων.

Τελικά, αυτά που πληροφορείται η σύζυγος από τον επιχειρηματία σχετικά με τους φίλους του είναι πως :

Ο Γιάννης και η Μαρία είναι παντρεμένοι. Ο Χρήστος είναι αθλητικός τύπος. Η Ελένη είναι προϊσταμένη σε τράπεζα.

Πράγματι, η σύζυγος φρόντισε και αγόρασε τα δώρα για τους φίλους του επιχειρηματία και τα έστειλε στο γραφείο του. Η παραλαβή των δώρων των φίλων του, έκρυβε για τον επιχειρηματία μια έκπληξη. Τα δώρα που παρέλαβε ήταν τρία, ενώ ο ίδιος περίμενε τέσσερα. Επικοινώνησε αμέσως με τη σύζυγό του, η οποία όμως τον διαβεβαίωσε, ότι έκανε τις επιλογές της ακριβώς σύμφωνα με τις πληροφορίες που της είχε ο ίδιος δώσει σχετικά με τους φίλους του. Το ένα από τα τρία δώρα προοριζόταν για το ζευγάρι των φίλων του. Στην πραγματικότητα όμως ζευγάρι φίλων δεν υπήρχε. Αμέσως έγινε κατανοητή η αιτία της παραπλάνησης.

Το πρόβλημα της αγοράς των δώρων αντιμετωπίστηκε από τη σύζυγο σύμφωνα με τις πληροφορίες που είχε πάρει. Η παραπλάνησή της οφείλεται στον τρόπο που ερμήνευσε την πρόταση *Ο Γιάννης και η Μαρία είναι παντρεμένοι.* Η πρόταση επιδέχεται δύο διαφορετικές, και ταυτόχρονα σωστές, ερμηνείες. Το γεγονός αυτό - της αποδεκτής διπλής ερμηνείας - οφείλεται στο ότι η διατύπωσή της αφήνει περιθώρια για κάτι τέτοιο.

Πρώτη ερμηνεία : *Ο Γιάννης και η Μαρία είναι παντρεμένοι μεταξύ τους.*

Δεύτερη ερμηνεία : *Ο Γιάννης είναι παντρεμένος και η Μαρία είναι παντρεμένη.*

Οι δύο διαφορετικές αυτές ερμηνείες οφείλονται στον ασαφή συνδετικό ρόλο που διαδραματίζει στη συγκεκριμένη πρόταση ο λογικός τελεστής ΚΑΙ. Δεν είναι σαφές από τη διατύπωση, αν ο τελεστής συνδέει δύο υποκείμενα μιας κύριας πρότασης (πρώτη ερμηνεία) ή αν συνδέει δύο υπονοούμενες κύριες προτάσεις (δεύτερη ερμηνεία).

Γίνεται λοιπόν αντιληπτό το ειδικό βάρος που έχει η σωστή διατύπωση στη σωστή κατανόηση ενός προβλήματος.

Σημαντικός ακόμα παράγοντας στη σωστή αντιμετώπιση ενός προβλήματος είναι η αποσαφήνιση του χώρου στον οποίο αναφέρεται. Η πληροφορία αυτή παρέχεται επίσης από την εκφώνηση του προβλήματος. Τα δεδομένα του προβλήματος είναι αυτά που θα μας παρέχουν αυτήν την πληροφορία.

Ορισμοί : Με τον όρο **δεδομένο** δηλώνεται οποιοδήποτε στοιχείο μπορεί να γίνει αντιληπτό από έναν τουλάχιστον παρατηρητή με μία από τις πέντε αισθήσεις του.

Με τον όρο **πληροφορία** αναφέρεται οποιοδήποτε γνωσιακό στοιχείο προέρχεται από επεξεργασία δεδομένων.

Ο όρος **επεξεργασία δεδομένων** δηλώνει εκείνη τη διαδικασία κατά την οποία ένας "μηχανισμός" δέχεται δεδομένα, τα επεξεργάζεται σύμφωνα με έναν προκαθορισμένο τρόπο και αποδίδει πληροφορίες.

Επί χιλιετίες ο "μηχανισμός" επεξεργασίας των δεδομένων ήταν και εξακολουθεί να είναι ο ανθρώπινος εγκέφαλος. Στις μέρες μας, ένας άλλος "μηχανισμός" επεξεργασίας δεδομένων είναι ο υπολογιστής.

1.3 Δομή προβλήματος

Η κατανόηση του προβλήματος είναι βασική προϋπόθεση για να γίνει στη συνέχεια δυνατή η σωστή αποτύπωση της δομής του. Η καταγραφή της δομής ενός προβλήματος σημαίνει αυτόματα ότι έχει αρχίσει η διαδικασία ανάλυσης του προβλήματος σε άλλα απλούστερα. Με τη σειρά τους τα νέα προβλήματα μπορούν να αναλυθούν σε άλλα, ακόμη πιο απλά. Η διαδικασία αυτή της ανάλυσης μπορεί να συνεχιστεί μέχρις ότου τα επιμέρους προβλήματα που προέκυψαν θεωρηθούν αρκετά απλά και η αντιμετώπισή τους χαρακτηριστεί ως δυνατή.

Ορισμός : Με τον όρο **δομή** ενός προβλήματος αναφερόμαστε στα συστατικά του μέρη, στα επιμέρους τμήματα που το αποτελούν καθώς επίσης και στον τρόπο που αυτά τα μέρη συνδέονται μεταξύ τους.

Η δυσκολία αντιμετώπισης των προβλημάτων ελαττώνεται όσο περισσότερο προχωράει η ανάλυση τους σε απλούστερα προβλήματα. Ο κατακερματισμός ενός προβλήματος σε άλλα απλούστερα είναι μια από τις διαδικασίες που ενεργοποιούν και αμβλύνουν τόσο τη σκέψη, αλλά κυρίως την αναλυτική ικανότητα του ατόμου.

Παράδειγμα 2

Ας υποθέσουμε ότι τίθεται σαν πρόβλημα το θέμα αντιμετώπισης των ναρκωτικών.

Η αντιμετώπιση του προβλήματος θα γίνει απλούστερη αν μπορέσουμε να αναλύσουμε το πρόβλημα σε άλλα απλούστερα Το αρχικό πρόβλημα είναι "Αντιμετώπιση ναρκωτικών". Αυτό θα μπορούσε να αναλυθεί καταρχήν σε τρία υποθέματα, σε τρία επιμέρους προβλήματα :

(1) Πρόληψη

(2) Θεραπεία

(3) Επανένταξη

Τα τρία αυτά επιμέρους προβλήματα πιθανό να μην είναι ιδιαίτερα λεπτομερή έτσι ώστε να επιτρέπουν την εύκολη αντιμετώπισή τους. Πρέπει λοιπόν κάθε ένα από αυτά να αναλυθεί σε ακόμα απλούστερα.

Έτσι λοιπόν το επί μέρους πρόβλημα (1) Πρόληψη, μπορεί να αναλυθεί σε :

(1.1) Σωστή ενημέρωση των πολιτών σχετικά με το θέμα

(1.2) Υποβοήθηση προς την κατεύθυνση ανάπτυξης ενδιαφερόντων, οραμάτων και στόχων εκ μέρους των εφήβων

(1.3) Υποστήριξη ομάδων αυξημένης θεωρητικά "προδιάθεσης"

Όμοια το επί μέρους πρόβλημα (2) Θεραπεία, μπορεί να αναλυθεί ως εξής :

(2.1) Δημιουργία νέων κρατικών θεραπευτικών κοινοτήτων

(2.2) Ενίσχυση υπαρχόντων θεραπευτικών κοινοτήτων

(2.3) Δημιουργία κατάλληλων τμημάτων στα δημόσια νοσοκομεία

Παρόμοια το επιμέρους πρόβλημα (3) Επανένταξη, μπορεί να αναλυθεί ως ακολούθως:

(3.1) Καταπολέμηση της κοινωνικής προκατάληψης έναντι των απεξαρτημένων

(3.2) Επιδότηση θέσεων εργασίας για απεξαρτημένους πρώην χρήστες

Στη συνέχεια και το πρόβλημα (1.1) μπορεί να αναλυθεί σε απλούστερα:

(1.1.1) Ενημέρωση των εφήβων μέσα από κατάλληλα προγράμματα στα σχολεία

(1.1.2) Ενημέρωση των γονέων με προγράμματα του Δήμου

(1.1.3) Ενημέρωση κάθε άλλου ενδιαφερόμενου πολίτη με προγράμματα του Υπουργείου Υγείας

Μια παρόμοια παραπέρα ανάλυση θα μπορούσε να γίνει και για το πρόβλημα (1.2), το οποίο θα μπορούσε να αναλυθεί στα εξής απλούστερα προβλήματα :

(1.2.1) Οργάνωση πολιτιστικών δραστηριοτήτων στα σχολεία

(1.2.2) Δημιουργία δημόσιων χώρων άθλησης στις γειτονιές για τους νέους

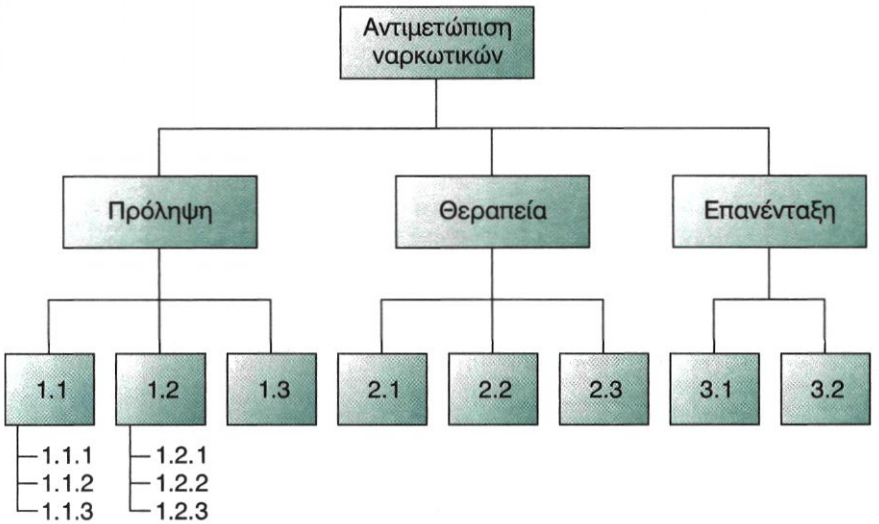
(1.2.3) Παροχή κινήτρων στα παιδιά και στους νέους για παρακολούθηση και συμμετοχή σε καλλιτεχνικά γεγονότα

Αν η ανάλυση του αρχικού προβλήματος θεωρείται επαρκής, η διάσπαση των επιμέρους προβλημάτων σε άλλα απλούστερα μπορεί να τερματιστεί. Ο παραπάνω τρόπος περιγραφής και ανάλυσης ενός προβλήματος γίνεται φραστικά. Ο ενδιαφερόμενος για την αντιμετώπιση του αρχικού προβλήματος, έχει πλέον μπροστά του να αντιμετωπίσει μια σειρά από επιμέρους προβλήματα, τα οποία στο σύνολό τους εκφράζουν και αντιστοιχούν στο αρχικό πρόβλημα.

Η ανάλυση αυτή του προβλήματος σε άλλα απλούστερα αναδύει παράλληλα και τη δομή του προβλήματος. Για τη γραφική απεικόνιση της δομής ενός προβλήματος χρησιμοποιείται συχνότατα η **διαγραμματική αναπαράσταση**. Σύμφωνα με αυτή:

- το αρχικό πρόβλημα αναπαρίσταται από ένα ορθογώνιο παραλληλόγραμμο
- κάθε ένα από τα απλούστερα προβλήματα στα οποία αναλύεται ένα οποιοδήποτε πρόβλημα, αναπαρίσταται επίσης από ένα ορθογώνιο παραλληλόγραμμο
- τα παραλληλόγραμμα που αντιστοιχούν στα απλούστερα προβλήματα στα οποία αναλύεται ένα οποιοδήποτε πρόβλημα, σχηματίζονται ένα επίπεδο χαμηλότερα. Έτσι σε κάθε κατώτερο επίπεδο, δημιουργείται η γραφική αναπαράσταση των προβλημάτων στα οποία αναλύονται τα προβλήματα του αμέσως ψηλότερου επιπέδου.

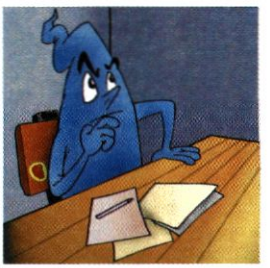
Η διαγραμματική αναπαράσταση του παραδείγματος που παρατίθεται προηγούμενα φαίνεται στο σχήμα 1.1.



Σχ. 1.1. *Διαγραμματική αναπαράσταση προβλήματος "Αντιμετώπιση ναρκωτικών"*

Η διαγραμματική αναπαράσταση προσφέρει μια απτή απεικόνιση της δομής του προβλήματος. Η δημιουργία του σχετικού διαγράμματος βοηθάει τόσο στην καλύτερη κατανόηση του ίδιου του προβλήματος, όσο και στη σχεδίαση της λύσης του.

1.4 Καθορισμός απαιτήσεων



Η σωστή επίλυση ενός προβλήματος προϋποθέτει τον επακριβή προσδιορισμό των **δεδομένων** που παρέχει το πρόβλημα. Απαιτεί επίσης την λεπτομερειακή καταγραφή των **ζητούμενων** που αναμένονται σαν αποτελέσματα της επίλυσης του προβλήματος.

Θα πρέπει να δοθεί μεγάλη προσοχή στην ανίχνευση των δεδομένων ενός προβλήματος. Επισημαίνεται πως δεν είναι πάντοτε εύκολο να διακρίνει κάποιος τα δεδομένα. Υπάρχουν πολλές περιπτώσεις προβλημάτων όπου τα δεδομένα θα πρέπει να "ανακαλυφθούν" μέσα στα λεγόμενα του προβλήματος. Η διαδικασία αυτή απαιτεί προσοχή, συγκέντρωση και σκέψη. Μεθοδολογία προσδιορισμού των δεδομένων ενός προβλήματος δεν υπάρχει, ούτε και μεθοδολογία εντοπισμού και αποσαφήνισης των ζητούμενων ενός προβλήματος.

Το ίδιο προσεκτικά θα πρέπει να αποσαφηνιστούν και τα ζητούμενα του προβλήματος. Δεν είναι πάντοτε ιδιαίτερα κατανοητό τι ακριβώς ζητάει ένα πρόβλημα. Σε μια τέτοια περίπτωση θα πρέπει να θέτονται μια σειρά από ερωτήσεις με στόχο την διευκρίνηση πιθανών απορριών σχετικά με τα ζητούμενα, τον τρόπο παρουσίασής τους, το εύρος τους κ.λπ. Οι ερωτήσεις αυτές μπορούν να απευθύνονται είτε στο δημιουργό του προβλήματος, είτε στον ίδιο μας τον εαυτό αν εμείς καλούμαστε να αντιμετωπίσουμε το πρόβλημα.

Παράδειγμα 3

Για λόγους αξιολόγησης της εκπαιδευτικής του πολιτικής, το Υπουργείο Παιδείας χρειάζεται να ενημερωθεί για τα πρόσφατα αποτελέσματα φοίτησης των μαθητών της χώρας. Ζήτησε λοιπόν μεταξύ άλλων, από την Υπηρεσία Πληροφορικής να παρουσιάσει και τα αποτελέσματα που είχαν οι μαθητές Γ' τάξης της Τεχνολογικής Κατεύθυνσης των Ενιαίων Λυκείων στα μαθήματα ειδικότητας.

Το πρόβλημα λοιπόν που τίθεται είναι : *Αποτελέσματα επίδοσης μαθητών Γ' τάξης Τεχνολογικής Κατεύθυνσης στα μαθήματα ειδικότητας.*

Το πρώτο πράγμα που απασχόλησε την Υπηρεσία Πληροφορικής του Υπουργείου ήταν να καταλάβει τι ακριβώς ζητείται, πράγμα που δεν φαίνεται αμέσως από το ιδιαίτερα σύντομο σημείωμα του προϊστάμενου. Έτσι προχώρησε στις εξής σκέψεις :

Προφανώς σε ό,τι αφορά στη χρονική περίοδο, πράγμα το οποίο δεν αναφέρεται, το Υπουργείο χρειάζεται τα αποτελέσματα της προηγούμενης σχολικής χρονιάς για όλη τη χώρα. Γνωρίζουμε σε πόσα και ποια Λύκεια λειτούργησε η Τεχνολογική Κατεύθυνση. Γνωρίζουμε επίσης ότι τα μαθήματα ειδικότητας της κατεύθυνσης αυτής είναι τέσσερα. Έτσι φαίνεται ότι ζητούνται τα αποτελέσματα φοίτησης όλων των μαθητών της Γ' τάξης των πιο πάνω Λυκείων για κάθε ένα από τα τέσσερα μαθήματα ειδικότητας.

Ποια είναι όμως ακριβώς τα ζητούμενα; Σε κάθε μάθημα κάθε μαθητής λαμβάνει έναν τελικό βαθμό. Αν συγκεντρώσουμε όλες τις βαθμολογίες από όλα τα σχολεία και για κάθε μάθημα, τότε αυτά είναι τα ζητούμενα αποτελέσματα;

Σε κάθε Λύκειο όλοι οι μαθητές της Γ' Λυκείου Τεχνολογικής Κατεύθυνσης διδάσκονται τα ίδια μαθήματα. Έτσι ότι ζητείται για το σύνολο των σχολείων, ζητείται καταρχήν για ένα σχολείο. Μπορούμε λοιπόν να μελετήσουμε το πρόβλημα για ένα σχολείο και μετά να το επεκτείνουμε για όλα τα Λύκεια της χώρας.

Πίνακας 1.1. Απόσπασμα βαθμολογίας μαθήματος		
ΣΧΟΛΕΙΟ : 9ο ΛΥΚΕΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ		
ΤΑΞΗ : Γ'		ΤΜΗΜΑ : 2
ΜΑΘΗΜΑ : Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον		
α/α	Ονοματεπώνυμο	Τελικός βαθμός
1	Μαϊκούσης Αθανάσιος	10
2	Μπουρνέλη Διονυσία	17
3	Μυλωνάς Αλέξανδρος	15

Κατόπιν αυτών η μελέτη επικεντρώνεται σε ένα σχολείο. Το σχολείο αυτό έχει 100 μαθητές στη Γ' τάξη Λυκείου Τεχνολογικής Κατεύθυνσης, κατανεμημένους σε τρία τμήματα. Οι βαθμολογίες των μαθητών και των τριών τμημάτων σε όλα τα μαθήματα επιλογής έχουν ήδη καταχωρηθεί. Ένα τμήμα της βαθμολογίας των μαθητών του πρώτου τμήματος για το μάθημα "Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον" φαίνεται στον πίνακα 1.1.

Υπάρχουν τρεις καταστάσεις βαθμολογίας, καθεμιά με τη βαθμολογία των μαθητών καθενός από τα τρία τμήματα. Αν τις ενώσουμε, θα έχουμε μία κατάσταση με τη βαθμολογία όλων των μαθητών της Γ' τάξης Τεχνολογικής Κατεύθυνσης του συγκεκριμένου σχολείου. Άρα αυτό είναι το ζητούμενο αποτέλεσμα.

Ωστόσο αμέσως μετά μπαίνουν τα εξής ερωτήματα : Μπορεί κανείς να βγάλει κάποιο συμπέρασμα διαβάζοντας μια λίστα με 100 ονόματα; Ακόμη και για να απαντηθεί το πιο απλό ερώτημα, έστω "πόσοι προάγονται", θα πρέπει κάποιος να τους μετρήσει. Βέβαια συμφέρει να μετρηθούν αυτοί που δεν προάγονται, οι οποίοι προφανώς θα είναι λιγότεροι, αλλά πάλι δεν αποφεύγεται η σάρωση όλης της λίστας. Θα πρέπει δε να γίνει παρόμοια εργασία για όλα τα σχολεία.

Ένα άλλο ερώτημα που τίθεται είναι το εξής: Για τους σκοπούς της ζητούμενης μελέτης ενδιαφέρει ποιοι ακριβώς μαθητές προάγονται και ποιοι όχι, ή μόνο πόσοι προάγονται και πόσοι όχι; Είναι φανερό ότι ενδιαφέρει το δεύτερο. Επομένως το αποτέλεσμα για το εξεταζόμενο σχολείο μπορεί να είναι ότι σε αυτό και στο μάθημα "Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον" προάγονται 92 μαθητές ενώ απορρίπτονται 8. Έχοντας κατά νου τον τελικό στόχο, δηλαδή όλη τη χώρα, θα μπορούσαμε να πάρουμε σαν τελικό αποτέλεσμα π.χ., ότι επί συνόλου 5809 μαθητών, 5287 μαθητές προάγονται και 522 απορρίπτονται στο συγκεκριμένο μάθημα. Α-

ντίστοιχοι αριθμοί θα υπάρχουν και για τα άλλα μαθήματα της κατεύθυνσης. Όμως μεγάλοι και πολλοί αριθμοί, αν και είναι ακριβείς, δεν παρέχουν αμέσως το ποιοτικό συμπέρασμα το οποίο είναι το ζητούμενο. Είναι προτιμότερο λοιπόν να τους εκφράσουμε σε ποσοστό, αφού έτσι γίνεται πιο εύκολα αντιληπτό το αποτέλεσμα.

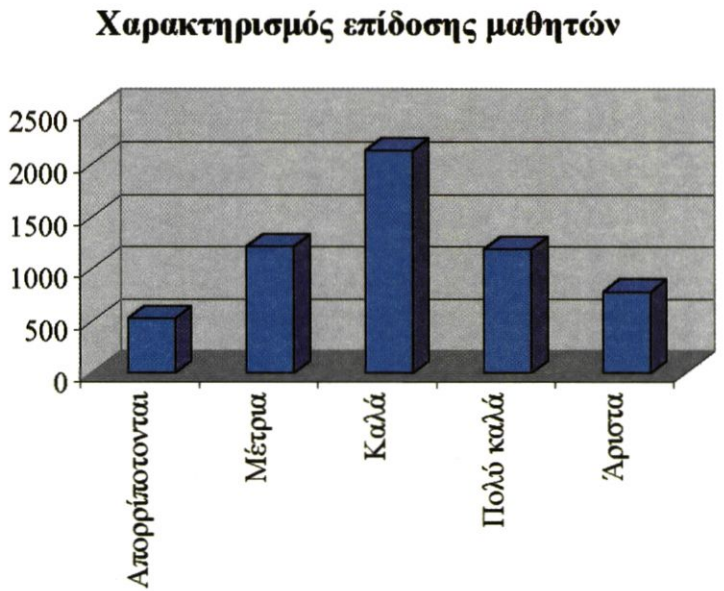
Όμως το προηγούμενο αποτέλεσμα δίνει έναν πρώτο χωρισμό των μαθητών σε επιτυχόντες και αποτυχόντες. Δεν απαντά στο ερώτημα ποιο ποσοστό μαθητών αρίστευσε, ποιο ποσοστό μαθητών πήρε βαθμολογία που κυμάνθηκε από 10 μέχρι 13 κ.λπ. Αν έχει κάποιος αυτής της μορφής τα αποτελέσματα, μπορεί να σχηματίσει πολύ καλύτερη εικόνα για την κατάσταση. Φαίνεται λοιπόν ότι είναι προτιμότερο τα ζητούμενα αποτελέσματα να περιλαμβάνουν και ποσοστά κατά κλίμακα βαθμολογίας, όπως περίπου παρουσιάζονται στον πίνακα 1.2.

Πίνακας 1.2. Αποτελέσματα στο μάθημα "Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον" σχολικού έτους 1999-2000.

Χαρακτηρισμός επίδοσης στο μάθημα	Βαθμός από	Βαθμός έως	Αριθμός μαθητών	Ποσοστό μαθητών %
Απορρίπτονται	0	9	522	9,0
Μέτρια	10	13	1211	20,8
Καλά	14	15	2120	36,5
Πολύ καλά	16	17	1180	20,3
Άριστα	18	20	776	13,4
ΣΥΝΟΛΟ			5809	100

Ο πίνακας 1.2 αποκαλείται στη Στατιστική πίνακας συχνοτήτων ή κατανομή συχνοτήτων, διότι παρουσιάζει το πλήθος των μονάδων ενός πληθυσμού που ανήκει σε ένα διάστημα τιμών (απόλυτη συχνότητα) και το ποσοστό των μονάδων του πληθυσμού που εντάσσεται στο ίδιο διάστημα τιμών (σχετική συχνότητα).

Μια άλλη σκέψη είναι ότι επειδή τα αποτελέσματα αυτά θα δημοσιοποιηθούν, θα ήταν καλύτερα αν απεικονιζόντουσαν γραφικά. Πράγματι με ένα γράφημα, όπως αυτό του σχήματος 1.2, είναι ο καθένας σε θέση με μια ματιά να αποκομίσει σημαντικές πληροφορίες, χωρίς να χρειάζεται την απομνημόνευση των αριθμών. Άλλωστε όπως λέει και μία παλιά κινέζικη παροιμία "μια εικόνα αξίζει όσο 1000 λέξεις".

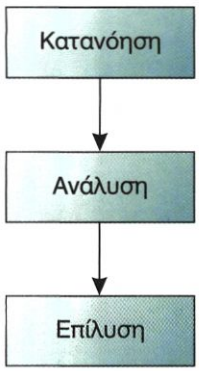


Σχ. 1.2. *Γράφημα ραβδογράμματος για το χαρακτηρισμό επίδοσης στο μάθημα.*

Τέλος μπορεί ακόμη να υπολογιστεί η μέση τιμή της βαθμολογίας, καθώς και η τυπική απόκλιση που δείχνει πόσο κατά μέσο όρο απέχουν οι βαθμολογίες από τη μέση τιμή.

Για την ολοκλήρωση της ανάλυσης του προβλήματος, θα πρέπει να γίνουν ανάλογες ενέργειες και για τα υπόλοιπα τρία μαθήματα ειδικότητας. Με παρόμοιο τρόπο θα πρέπει να υπολογιστούν καταρχήν οι επιδόσεις των μαθητών της Γ' τάξης Λυκείου Τεχνολογικής Κατεύθυνσης του ενός σχολείου και στη συνέχεια όλων των σχολείων της χώρας, δημιουργώντας και για αυτά τα μαθήματα τους αντίστοιχους πίνακες και τα κατάλληλα γραφήματα.

Εδώ η Υπηρεσία Πληροφορικής κρίνει ότι έχει ολοκληρώσει την ανάλυση του προβλήματος, που της ζητήθηκε να λύσει. Θεωρείται ότι το πρόβλημα έχει κατανοηθεί πλήρως. Είναι γνωστό πλέον το περιβάλλον του προβλήματος, πρώτα στο επίπεδο σχολείου και κατόπιν στο επίπεδο όλης της χώρας, ενώ η πηγή των δεδομένων είναι οι καταστάσεις βαθμολογιών, οι οποίες υπάρχουν. Τα ζητούμενα αποτελέσματα έχουν αποσαφηνιστεί, και η μορφή που παρέχονται είναι τόσο αριθμητική (απόλυτη και ποσοστιαία), όσο και διαγραμματική.



Σχ. 1.3. Στάδια αντιμετώπισης προβλήματος

Συμπερασματικά από όλα τα παραπάνω διαφαίνεται πως τα στάδια αντιμετώπισης ενός προβλήματος είναι τρία (σχήμα 1.3):

- κατανόηση, όπου απαιτείται η σωστή και πλήρης αποσαφήνιση των δεδομένων και των ζητούμενων του προβλήματος
- ανάλυση, όπου το αρχικό πρόβλημα διασπάται σε άλλα επί μέρους απλούστερα προβλήματα
- επίλυση, όπου υλοποιείται η λύση του προβλήματος, μέσω της λύσης των επιμέρους προβλημάτων.

1.5 Κατηγορίες προβλημάτων

Τα προβλήματα που απαντώνται τόσο στους διάφορους επιστημονικούς τομείς, όσο και στην καθημερινή μας ζωή, ποικίλουν ως προς τη φύση τους. Από τα παραδείγματα που έχουμε παραθέσει, έχει γίνει αντιληπτό πως τα προβλήματα δεν σχετίζονται υποχρεωτικά και αποκλειστικά με τα μαθηματικά ή γενικότερα με μαθηματικές και υπολογιστικές διαδικασίες με σκοπό την επίτευξη λύσης τους. Η διαφορετική φύση των προβλημάτων επιτρέπει την κατηγοριοποίησή τους σύμφωνα με ποικίλα κριτήρια.

1. Με κριτήριο τη δυνατότητα επίλυσης ενός προβλήματος, διακρίνουμε τρεις κατηγορίες προβλημάτων :

- **Επιλύσιμα**, είναι εκείνα τα προβλήματα για τα οποία η λύση τους είναι ήδη γνωστή και έχει διατυπωθεί. Επιλύσιμα μπορεί επίσης να χαρακτηριστούν και προβλήματα, των οποίων η λύση δεν έχει ακόμα διατυπωθεί, αλλά ή συνάφειά τους με άλλα ήδη επιλυμένα μας επιτρέπει να θεωρούμε σαν βέβαιη τη δυνατότητα επίλυσής τους.
- **Ανοικτά**, ονομάζονται εκείνα τα προβλήματα για τα οποία η λύση τους δεν έχει μεν ακόμα βρεθεί, αλλά παράλληλα δεν έχει αποδειχθεί, ότι δεν επιδέχονται λύση. Σαν παράδειγμα ανοικτού προβλήματος μπορούμε να αναφέρουμε το πρόβλημα της ενοποίησης των τεσσάρων πεδίων δυνάμεων, που αναφέρουμε σε προηγούμενη παράγραφο.
- **Άλυτα**, χαρακτηρίζονται εκείνα τα προβλήματα για τα οποία έχουμε φτάσει στην παραδοχή, ότι δεν επιδέχονται λύση. Τέτοιου είδους πρόβλημα είναι το γνωστό από τους αρχαίους ελληνικούς χρόνους πρόβλημα του τετραγωνισμού του κύκλου. Το πρόβλημα αυτό θεωρείται άλυτο, στην πραγματικότητα η λύση που επιδέχεται είναι προσεγγιστική.

2. Με κριτήριο το βαθμό δόμησης των λύσεών τους, τα επιλύσιμα προβλήματα μπορούν να διακριθούν σε τρεις επίσης κατηγορίες :

- **Δομημένα**, χαρακτηρίζονται εκείνα τα προβλήματα των οποίων η επίλυση προέρχεται από μια αυτοματοποιημένη διαδικασία. Για παράδειγμα, η επίλυση της δευτεροβάθμιας εξίσωσης αποτελεί ένα δομημένο πρόβλημα, αφού ο τρόπος επίλυσης της εξίσωσης είναι γνωστός και αυτοματοποιημένος.

- **Ημιδομημένα**, ονομάζονται τα προβλήματα εκείνα των οποίων η λύση επιδιώκεται στα πλαίσια ενός εύρους πιθανών λύσεων, αφήνοντας στον ανθρώπινο παράγοντα περιθώρια επιλογής της. Σαν παράδειγμα ημιδομημένου προβλήματος μπορούμε να αναφέρουμε ένα πρόβλημα όπου ένας ταξιδιώτης αναζητά να επιλέξει το μεταφορικό μέσο μετακίνησής του από ένα μέρος σε κάποιο άλλο. Το πρόβλημα είναι ημιδομημένο, δεδομένου ότι η λύση που θα επιλεγεί, πρέπει να αναζητηθεί σε ένα σύνολο σαφώς προκαθορισμένο που συμπεριλαμβάνει όλα τα διαθέσιμα μεταφορικά μέσα.

- **Αδόμητα**, χαρακτηρίζονται τα προβλήματα εκείνα των οποίων οι λύσεις δεν μπορούν να δομηθούν ή δεν έχει διερευνηθεί σε βάθος η δυνατότητα δόμησής τους. Πρωτεύοντα ρόλο στην επίλυση αυτού του τύπου προβλημάτων κατέχει η ανθρώπινη διαίσθηση. Παράδειγμα αδόμητου προβλήματος είναι η επιλογή του τρόπου, του τόπου και του χρόνου ενός εφηβικού πάρτυ. Είναι σαφές ότι δεν υπάρχει κανένας προδιατυπωμένος τρόπος οργάνωσης ενός εφηβικού πάρτυ και όλοι οι παράγοντες που θα το διαμορφώσουν επαφίονται στην ανθρώπινη αίσθηση και προτίμηση των διοργανωτών του.

3. Το κάθε πρόβλημα σε ότι αφορά στην επίλυσή του, είναι στενά συνδεδεμένο με την έννοια του αλγόριθμου που παρουσιάζουμε αναλυτικά στο επόμενο κεφάλαιο. Με κριτήριο το είδος της επίλυσης που επιζητούν, τα προβλήματα διακρίνονται σε τρεις κατηγορίες :

- **Απόφασης**, όπου η απόφαση που πρόκειται να ληφθεί σαν λύση του προβλήματος που τίθεται, απαντά σε ένα ερώτημα και πιθανόν αυτή η απάντηση να είναι ένα "Ναι" ή ένα "Όχι". Αυτό που θέλουμε να διαπιστώσουμε σε ένα πρόβλημα απόφασης είναι αν υπάρχει απάντηση που ικανοποιεί τα δεδομένα που θέτονται από το πρόβλημα.

Παράδειγμα: Δίδεται ένας ακέραιος αριθμός Ν και το πρόβλημα που τίθεται είναι, αν ο αριθμός Ν είναι πρώτος.

- **Υπολογιστικά**, όπου το πρόβλημα που τίθεται απαιτεί τη διενέργεια υ-

Με μια απλοποιημένη προσέγγιση, αλγόριθμος είναι μια "συνταγή" που προσδιορίζει τι πρέπει να κάνουμε κάτω από ορισμένες συνθήκες, έτσι ώστε να φτάσουμε στον επιθυμητό σκοπό.

πολογισμών, για να μπορεί να δοθεί μία απάντηση στο πρόβλημα. Σε ένα υπολογιστικό πρόβλημα ζητάμε να βρούμε τη τιμή της απάντησης που ικανοποιεί τα δεδομένα που παρέχει το πρόβλημα.

Παράδειγμα: Δίδεται ένας ακέραιος αριθμός Ν και ζητείται να βρεθεί πόσες διαφορετικές παραγοντοποιήσεις του Ν υπάρχουν.

- **Βελτιστοποίησης**, όπου το πρόβλημα που τίθεται επιζητά το βέλτιστο αποτέλεσμα για τα συγκεκριμένα δεδομένα που διαθέτει. Σε ένα πρόβλημα βελτιστοποίησης αναζητούμε την απάντηση που ικανοποιεί κατά τον καλύτερο τρόπο τα δεδομένα που παρέχει το πρόβλημα.

Παράδειγμα: Δίδεται ένας ακέραιος αριθμός Ν και ζητείται ποια είναι η παραγοντοποίηση για το Ν με το μεγαλύτερο πλήθος παραγόντων.

1.6 Πρόβλημα και υπολογιστής

Τα προβλήματα που καθημερινά συναντάμε δεν είναι προβλήματα του χώρου των μαθηματικών, της φυσικής ή της στατιστικής. Συνήθως είναι λιγότερο ή περισσότερο σύνθετα προβλήματα που η αντιμετώπισή τους γίνεται αποκλειστικά και μόνο με βάση τους συλλογισμούς μας και τη σκέψη μας. Σε κάποιες ίσως περιπτώσεις χρειάζεται να πιάσουμε μολύβι και χαρτί, ή έστω και κάποιο κομπιουτεράκι, για να κάνουμε μερικούς αναγκαίους υπολογισμούς. Όμως οι όποιες προσπάθειες αντιμετώπισης προβλημάτων στηρίζονται στις νοητικές μας δυνάμεις, στη συλλογιστική μας και στη ικανότητα λήψης αποφάσεων.

Και οι υπολογιστές που έχουν κυριολεκτικά εισβάλλει στη ζωή μας, που καθημερινά ερχόμαστε σε επαφή μαζί τους, που αναντίρρητα εφαρμόζονται σε ένα μεγάλο εύρος τομέων και δραστηριοτήτων, σχετίζονται με την επίλυση προβλημάτων, και αν ναι με ποιο τρόπο;

Αναμφίβολα προβλήματα λυνόντουσαν και πριν τη "γένεση" των υπολογιστών. Η ικανότητα του ανθρώπου να αντιμετωπίζει και να επιλύει προβλήματα είναι πολύ προγενέστερη της εμφάνισής τους. Οι υπολογιστές ήρθαν σχετικά πάρα πολύ πρόσφατα για να δράσουν επικουρικά στην ανθρώπινη δραστηριότητα.

Η ένα προς ένα αντιπαράθεση στοιχείων μεταξύ ανθρώπου και υπολογιστή είναι ένα ενδιαφέρον "παιγνίδι". Η Πληροφορική δεν έχει να κάνει πλέον μόνο με τους υπολογιστές, αλλά με τον τρόπο ζωής του ανθρώπου. Το ανθρώπινο είδος, νοήμον ον, διαθέτει έμφυτες όλες εκείνες τις ικανότη-

τες που απαιτούνται για την επιτυχή αντιμετώπιση μιας τεράστιας γκάμας προβλημάτων. Η πολυπλοκότητα των συλλογισμών που μπορεί να εκτελεί ο ανθρώπινος εγκέφαλος είναι εξαιρετικά μεγάλη. Ο ακριβής τρόπος λειτουργίας του ανθρώπινου εγκεφάλου εξακολουθεί να παραμένει ένα αναπάντητο, τουλάχιστον προς το παρόν, ερώτημα.

Η "σύγκριση" λειτουργιών ανθρώπου και υπολογιστή επιφέρει βέβαια μια τεράστια ποιοτική διαφορά υπέρ του ανθρώπου. Ο υπολογιστής δεν είναι ένας ηλεκτρονικός εγκέφαλος. Αυτό που κάνει δεν είναι τίποτε περισσότερο από το να χειρίζεται στοιχεία, ενώ το ανθρώπινο πνεύμα μπορεί να σκέπτεται, να παράγει ιδέες. Το σημείο αυτό είναι πρωταρχικής σημασίας, προσδιορίζοντας μια αναμφισβήτητα τεράστια ποιοτική διαφορά. Το σημείο εκείνο στο οποίο ο υπολογιστής υπερτερεί έναντι του ανθρώπου, είναι η ταχύτητα εκτέλεσης των πράξεών του, ταχύτητα η οποία βελτιώνεται συνέχεια κατά αλματώδη τρόπο με την πρόοδο της τεχνολογίας.

Προβλήματα τα οποία απαιτούν πολλούς υπολογισμούς για την αντιμετώπισή τους, ενδείκνυνται για ανάθεση προς επίλυση σε υπολογιστή. Από τα παραδείγματα που αναφέρθηκαν στο κεφάλαιο, αυτό το παράδειγμα του οποίου η επίλυση θα μπορούσε να ανατεθεί σε υπολογιστή είναι το παράδειγμα 3 σχετικά με τα αποτελέσματα φοίτησης των μαθητών.

Οι λόγοι που αναθέτουμε την επίλυση ενός προβλήματος σε υπολογιστή σχετίζονται με

- ✓ την πολυπλοκότητα των υπολογισμών,
- ✓ την επαναληπτικότητα των διαδικασιών,
- ✓ την ταχύτητα εκτέλεσης των πράξεων,
- ✓ το μεγάλο πλήθος των δεδομένων.

Όσο και αν τυχόν ξαφνιάζει, ο υπολογιστής δεν μπορεί να εκτελεί παρά μόνο τρεις λειτουργίες :

- ✓ πρόσθεση, η οποία αποτελεί τη βασική αριθμητική πράξη, δεδομένου ότι και οι άλλες αριθμητικές πράξεις μπορούν να αντιμετωπιστούν, σαν διαδικασίες πρόσθεσης
- ✓ σύγκριση, η οποία συνιστά τη βασική λειτουργία για την επιτέλεση όλων των λογικών πράξεων,
- ✓ μεταφορά δεδομένων, λειτουργία που προηγείται και έπεται της επεξεργασίας δεδομένων.

Οι λειτουργίες αυτές είναι αρκετές, ώστε ο υπολογιστής να επιτελέσει με επιτυχία κάθε είδους επεξεργασία. Με βάση αυτές τις τρεις λειτουργίες

διεκπεραιώνει όλες τις εργασίες που του αναθέτονται και επιλύει όλα τα προβλήματα που αναλαμβάνει.

Είναι σαφές ότι ο άνθρωπος θα χρειαζόταν χρόνια ή και αιώνες για να εκτελέσει τις πράξεις εκείνες που ο υπολογιστής μπορεί να τις ολοκληρώσει μέσα σε λίγα μόλις λεπτά. Είναι λοιπόν ιδιαίτερα χρήσιμη και σημαντική η προσφορά του. Όμως δεν θα πρέπει να ξεχνάει κανείς πως η ικανότητα που παρουσιάζει ο υπολογιστής εκδηλώνεται σε ποσοτικό επίπεδο και όχι σε ποιοτικό. Μπορεί να αντιμετωπίσει σύνθετα λογικά προβλήματα μόνο εφόσον ο άνθρωπος έχει φροντίσει προηγούμενα, με τη χρήση κατάλληλων προγραμμάτων, να του "διδάξει" τον τρόπο αντιμετώπισης και επίλυσης αυτού του είδους των προβλημάτων. Πρέπει να γίνει απόλυτα κατανοητό πως τα προβλήματα και οι λύσεις του προϋπήρξαν και εξακολουθούν να υπάρχουν ανεξάρτητα από τους υπολογιστές.

Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάστηκε η διαχρονικότητα του προβλήματος και έγινε σαφής η ανεξαρτησία της λύσης του από τον υπολογιστή. Επισημάνθηκαν βασικά στοιχεία στην ανάλυση και σύνταξη προβλημάτων που αφορούν στη σαφήνεια της διατύπωσης και κατ' επέκταση στην κατανόηση του προβλήματος. Στη συνέχεια παρουσιάστηκε η έννοια της δομής ενός προβλήματος, που ουσιαστικά ανάγει την ανάλυσή του σε άλλα απλούστερα. Ο καθορισμός δεδομένων και ζητούμενων ήταν αυτό που μας απασχόλησε στη συνέχεια. Ολοκληρώνοντας το πρώτο αυτό κεφάλαιο παρουσιάσαμε διαφορετικές κατηγοριοποιήσεις των προβλημάτων. Τέλος προσδιορίσαμε τους λόγους που συνηγορούν υπέρ της ανάθεσης προβλήματος σε υπολογιστή, εντοπίζοντας ταυτόχρονα τις ποιοτικές διαφορές αντιμετώπισης προβλημάτων ανάμεσα στον υπολογιστή και τον άνθρωπο.

Λέξεις κλειδιά

Δεδομένο, διαγραμματική αναπαράσταση, δομή προβλήματος, επεξεργασία δεδομένων, κατηγορίες προβλημάτων, πληροφορία, πρόβλημα.

Ερωτήσεις - Θέματα για συζήτηση

- Να γίνει συζήτηση σχετικά με την αντιμετώπιση προβλημάτων με τη χρήση υπολογιστών.
- Να γίνει συζήτηση σχετικά με τη δημιουργία προβλημάτων εξ αιτίας της χρήσης υπολογιστών.
- Οι μαθητές να αναφέρουν ιεραρχώντας τα σημαντικότερα κοινωνικά προβλήματα που κατά τη γνώμη τους σχετίζονται με την επιστήμη της Πληροφορικής.
- Να δοθεί ο ορισμός των όρων δεδομένο, επεξεργασία δεδομένων, πληροφορία.
- Να γίνει η γραφική αναπαράσταση του προβλήματος που περιγράφεται στο παράδειγμα της ενότητας 1.4.
- Να αναφερθούν οι κατηγορίες των προβλημάτων.
- Για ποιους λόγους αναθέτεται η επίλυση ενός προβλήματος σε υπολογιστή;
- Περιγράψτε τους τρόπους περιγραφής και αναπαράστασης των προβλημάτων.

Βιβλιογραφία

1. Jacques Arsac, Les machines á penser Des ordinateurs et des hommes, Seuil, Paris, 1987.
2. Emanuel Falkenauer, Genetic algorithms and grouping problems, Wiley, 1998.
3. Les Goldschlager & Andrew Lister: Computer Science A modern introduction, Prentice Hall, 1990.
4. R. Kadesch, Problem Solving-Across the Disciplines, Prentice Hall Engineering, Science & Math, 1996.
5. Nicholas Negroponte, Being digital, Alfred Knopf Inc, 1995.
6. G. Polya, How to solve it A new aspect of mathematical method, Princeton University Press (second edition renewed), 1985.

Διευθύνσεις Διαδικτύου

- http://www.awesomelibrary.org/Classroom/Science/Problem_Solving/Problem_Solving.html

Ιστοσελίδα με αρκετά ενδιαφέρουσες απόψεις σε ότι αφορά στην επίλυση προβλημάτων. Αρκετά υψηλού επιπέδου θεματολογίας και σχετικής δυσκολίας, προτείνεται για μαθητές με αυξημένο ενδιαφέρον.

- http://www.hawaii.edu/suremath/click.html

Ιστοσελίδα με ιδιαίτερα χαρούμενο και νεανικό περιβάλλον, αναφέρεται στην επίλυση προβλήματος και περιλαμβάνει, εκτός από τη θεωρία περί επίλυσης προβλήματος, μια σειρά από προβλήματα που προέρχονται από διαφορετικά σχολικά μαθήματα (άλγεβρα, φυσική, χημεία) και άλλους τομείς δραστηριοτήτων.

ΚΕΦΑΛΑΙΟ 2ο

ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΑΛΓΟΡΙΘΜΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ

- Τι είναι αλγόριθμος
- Σπουδαιότητα αλγορίθμων
- Περιγραφή και αναπαράσταση αλγορίθμων
- Βασικές συνιστώσες/εντολές ενός αλγορίθμου

Εισαγωγή

Αρχικά εξηγείται ο όρος αλγόριθμος και παραθέτονται τα σπουδαιότερα κριτήρια που πρέπει να πληροί κάθε αλγόριθμος. Στη συνέχεια, η σπουδαιότητα των αλγορίθμων συνδυάζεται με την εξέλιξη της επιστήμης της Πληροφορικής. Η περιγραφή και αναπαράσταση των αλγορίθμων δίνεται αναλυτικά με χρήση των μεθόδων αναπαράστασης ελεύθερου κειμένου, διαγραμμάτων ροής, φυσικής γλώσσας και κωδικοποίησης με πρόγραμμα. Τα προγράμματα παρουσιάζονται με τη μορφή ψευδογλώσσας, που ορίζεται και τυποποιείται σε ένα σύνολο εντολών και προγραμματιστικών ακολουθιακών ενοτήτων. Στη συνέχεια, δίνονται παραδείγματα όπου εξετάζονται οι διάφορες συνιστώσες ενός αλγόριθμου, δηλαδή οι απαραίτητες εντολές που στηρίζουν το 'κτίσιμο' ενός αλγόριθμου. Συγκεκριμένα, παρουσιάζονται η δομή ακολουθίας, η δομή της επιλογής, οι επαναληπτικές διαδικασίες, οι διαδικασίες πολλαπλών επιλογών και οι εμφωλιασμένες διαδικασίες. Για κάθε τύπο συνιστώσας δίνονται αναλυτικά παραδείγματα σε φυσική γλώσσα, σε ακολουθία διαδοχικών βημάτων και σε μορφή διαγραμμάτων ροής. Στο τέλος του κεφαλαίου παρουσιάζεται η ανάπτυξη και η αλγοριθμική προσέγγιση για την επίλυση ενός συνθετότερου προβλήματος , του προβλήματος του 'πολλαπλασιασμού αλά ρωσικά', όπου γίνεται χρήση και συνδυασμός αλγοριθμικών συνιστωσών.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να διατυπώνουν την έννοια του αλγορίθμου,
- να αιτιολογούν τη σπουδαιότητα των αλγορίθμων,
- να τεκμηριώνουν την αναγκαιότητα της αλγοριθμικής προσέγγισης κατά τη διαδικασία επίλυσης προβλημάτων,
- να εφαρμόζουν τυποποιημένη επίλυση με αλγοριθμικές διαδικασίες,
- να μπορούν να σχεδιάζουν αλγόριθμους με χρήση συγκεκριμένων τεχνικών.

Προερωτήσεις

- ✓ Γνωρίζεις τι είναι αλγοριθμική προσέγγιση;
- ✓ Ξέρεις ότι ήδη έχεις χρησιμοποιήσει πολλούς αλγορίθμους;
- ✓ Γνωρίζεις, αν ο πολλαπλασιασμός δύο αριθμών μπορεί να γίνει με άλλο τρόπο;
- ✓ Τι θα κάνεις για να βρεις το άθροισμα 3+6+9+...+999;

2.1 Τι είναι αλγόριθμος

Η θεωρία των αλγορίθμων έχει μεγάλη παράδοση και η ηλικία μερικών αλγορίθμων αριθμεί χιλιάδες χρόνια, όπως για παράδειγμα ο αλγόριθμος του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη δύο αριθμών ή το λεγόμενο κόσκινο του Ερατοσθένη για την εύρεση των πρώτων αριθμών από 1 ως n. Σήμερα το πεδίο της Θεωρίας Αλγορίθμων είναι ένα ιδιαίτερα ευρύ και πλούσιο πεδίο. Πληθώρα συγγραμμάτων έχει εμφανισθεί στη βιβλιογραφία, ενώ συνεχίζεται η περαιτέρω εμβάθυνση σε νέα σύγχρονα προβλήματα. Οι περισσότεροι από τους αλγορίθμους που συνήθως εξετάζονται στα σχετικά βιβλία έχουν προταθεί τα τελευταία 25 χρόνια, όση περίπου είναι και η ηλικία της Πληροφορικής ως μίας νέας αυθύπαρκτης επιστήμης.

Ιστορικό σημείωμα

Η λέξη *αλγόριθμος* (algorithm) προέρχεται από μια μελέτη του Πέρση μαθηματικού Abu Ja'far Mohammed ibn Musa al Khowarizmi, που έζησε περί το 825 μ.Χ. Πέντε αιώνες αργότερα η μελέτη αυτή μεταφράστηκε στα λατινικά και άρχιζε με τη φράση "Algoritmi dixit ..." (ο αλγόριθμος λέει ....). Η μελέτη του al Khowarizmi υπήρξε η πρώτη πλήρης πραγματεία άλγεβρας (όρος που και αυτός προέρχεται από το αραβικό al-jabr=αποκατάσταση), γιατί ένας από τους σκοπούς της άλγεβρας είναι και η αποκατάσταση της ισότητας μέσα σε μια εξίσωση. Ο όρος αλγόριθμος επέζησε επί χίλια χρόνια ως σπάνιος όρος, που σήμαινε κάτι σαν "συστηματική διαδικασία αριθμητικών χειρισμών". Τη σημερινή του αξία απόκτησε από την αρχή του 20ού αιώνα με την ανάπτυξη της ομώνυμης θεωρίας και φυσικά με την επικαιρότητα των ηλεκτρονικών υπολογιστών.

Ο όρος αλγόριθμος, λοιπόν, χρησιμοποιείται για να δηλώσει μεθόδους που εφαρμόζονται για την επίλυση προβλημάτων. Ωστόσο, ένας πιο αυστηρός ορισμός της έννοιας αυτής είναι ο εξής.

Ορισμός: *Αλγόριθμος είναι μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος.*

Κάθε αλγόριθμος απαραίτητα ικανοποιεί τα επόμενα κριτήρια.

- ***Είσοδος*** (input). Καμία, μία ή περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δεν δίνονται τιμές δεδομένων εμφανίζεται, όταν ο αλγόριθμος δημιουργεί και επεξερ-

γάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια συναρτήσεων παραγωγής τυχαίων αριθμών ή με τη βοήθεια άλλων απλών εντολών.

- ***Έξοδος*** (output). Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή δεδομένων ως αποτέλεσμα προς το χρήστη ή προς έναν αλλο αλγόριθμο.
- ***Καθοριστικότητα*** (definiteness). Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της. Λόγου χάριν, μία εντολή διαίρεσης πρέπει να θεωρεί και την περίπτωση, όπου ο διαιρέτης λαμβάνει μηδενική τιμή.
- ***Περατότητα*** (finiteness). Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα συγκεκριμένο αριθμό βημάτων δεν αποτελεί αλγόριθμο, αλλά λέγεται απλά *υπολογιστική διαδικασία* (computational procedure).
- ***Αποτελεσματικότητα*** (effectiveness). Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή. Αυτό σημαίνει ότι μία εντολή δεν αρκεί να έχει ορισθεί, αλλά πρέπει να είναι και εκτελέσιμη.

Η έννοια του αλγόριθμου δεν συνδέεται αποκλειστικά και μόνο με προβλήματα της Πληροφορικής. Ας θεωρήσουμε, για παράδειγμα, ότι θέλουμε να γευματίσουμε και επομένως πρέπει να εκτελέσουμε τις επόμενες ενέργειες:

- ✓ να συγκεντρώσουμε τα υλικά,
- ✓ να προετοιμάσουμε τα σκεύη μαγειρικής,
- ✓ να παρασκευάσουμε το φαγητό,
- ✓ να ετοιμάσουμε τη σαλάτα,
- ✓ να στρώσουμε το τραπέζι,
- ✓ να γευματίσουμε,
- ✓ να καθαρίσουμε το τραπέζι, και
- ✓ να πλύνουμε τα πιάτα και τα κουζινικά.

Είναι ευνόητο ότι η προηγούμενη αλληλουχία των ενεργειών οδηγεί στο επιθυμητό αποτέλεσμα. Βέβαια, αυτή η αλληλουχία δεν είναι η μοναδική για την επίτευξη του σκοπού, αφού, για παράδειγμα, μπορούμε πρώτα να ετοιμάσουμε τη σαλάτα και μετά να παρασκευάσουμε το φαγητό, ενώ ακόμη μπορούμε πρώτα να πλύνουμε τα πιάτα και μετά να καθαρίσουμε το τραπέζι. Ωστόσο, το παράδειγμα θέλει να δείξει, ότι η θεώρηση μίας σύνθε-

της εργασίας με διακριτά βήματα που εκτελούνται διαδοχικά, είναι ένας πολύ χρήσιμος και πρακτικός τρόπος σκέψης για την επίλυση πολλών (αν όχι όλων) προβλημάτων.

2.2 Σπουδαιότητα αλγορίθμων

Η έννοια του αλγόριθμου είναι θεμελιώδης για την επιστήμη της Πληροφορικής. Η μελέτη των αλγορίθμων είναι πολύ ενδιαφέρουσα, γιατί είναι η πρώτη ύλη για τη μελέτη και εμβάθυνση, αν όχι σε όλες, τουλάχιστον σε πάρα πολλές γνωστικές περιοχές της επιστήμης αυτής.

Η Πληροφορική, λοιπόν, μπορεί να ορισθεί ως η επιστήμη που μελετά τους αλγορίθμους από τις ακόλουθες σκοπιές:

- ***Υλικού*** (hardware). Η ταχύτητα εκτέλεσης ενός αλγορίθμου επηρεάζεται από τις διάφορες τεχνολογίες υλικού, δηλαδή από τον τρόπο που είναι δομημένα σε μία ενιαία αρχιτεκτονική τα διάφορα συστατικά του υπολογιστή (δηλαδή ανάλογα με το αν ο υπολογιστής έχει κρυφή μνήμη και πόση, ανάλογα με την ταχύτητα της κύριας και δευτερεύουσας μνήμης κοκ.).

- ***Γλωσσών Προγραμματισμού*** (programming languages). Το είδος της γλώσσας προγραμματισμού που χρησιμοποιείται (δηλαδή, χαμηλότερου ή υψηλότερου επιπέδου) αλλάζει τη δομή και τον αριθμό των εντολών ενός αλγορίθμου. Γενικά μία γλώσσα που είναι χαμηλοτέρου επιπέδου (όπως η assembly ή η γλώσσα C) είναι ταχύτερη από μία άλλη γλώσσα που είναι υψηλοτέρου επιπέδου (όπως η Basic ή Pascal). Ακόμη, σημειώνεται ότι διαφορές συναντώνται μεταξύ των γλωσσών σε σχέση με το πότε εμφανίσθηκαν. Για παράδειγμα, παλαιότερα μερικές γλώσσες προγραμματισμού δεν υποστήριζαν την αναδρομή (έννοια που θα εξετάσουμε σε βάθος αργότερα).

- ***Θεωρητική*** (theoretical). Το ερώτημα που συχνά τίθεται είναι, αν πράγματι υπάρχει ή όχι κάποιος αποδοτικός αλγόριθμος για την επίλυση ενός προβλήματος. Η εξέταση αυτού του ερωτήματος είναι δύσκολο να σχολιασθεί στα πλαίσια του βιβλίου αυτού, επειδή απαιτεί μεγάλη θεωρητική κατάρτιση. Ωστόσο η προσέγγιση αυτή είναι ιδιαίτερα σημαντική, γιατί προσδιορίζει τα όρια της λύσης που θα βρεθεί σε σχέση με ένα συγκεκριμένο πρόβλημα.

- ***Αναλυτική*** (analytical). Μελετώνται οι υπολογιστικοί πόροι (computer resources) που απαιτούνται από έναν αλγόριθμο, όπως για παράδειγ-

μα το μέγεθος της κύριας και της δευτερεύουσας μνήμης, ο χρόνος για λειτουργίες CPU και για λειτουργίες εισόδου/εξόδου κ.λπ. Το αντικείμενο αυτό θα εξηγηθεί πληρέστερα στο Κεφάλαιο 5.

2.3 Περιγραφή και αναπαράσταση αλγορίθμων

Στη βιβλιογραφία συναντώνται διάφοροι τρόποι αναπαράστασης ενός αλγορίθμου:

- με ***ελεύθερο κείμενο*** (free text), που αποτελεί τον πιο ανεπεξέργαστο και αδόμητο τρόπο παρουσίασης αλγορίθμου. Έτσι εγκυμονεί τον κίνδυνο ότι μπορεί εύκολα να οδηγήσει σε μη εκτελέσιμη παρουσίαση παραβιάζοντας το τελευταίο χαρακτηριστικό των αλγορίθμων, δηλαδή την αποτελεσματικότητα.
- με ***διαγραμματικές τεχνικές*** (diagramming techniques), που συνιστούν ένα γραφικό τρόπο παρουσίασης του αλγορίθμου. Από τις διάφορες διαγραμματικές τεχνικές που έχουν επινοηθεί, η πιο παλιά και η πιο γνωστή ίσως, είναι το διάγραμμα ροής (flow chart). Ωστόσο η χρήση διαγραμμάτων ροής για την παρουσίαση αλγορίθμων δεν αποτελεί την καλύτερη λύση, γι'αυτό και εμφανίζονται όλο και σπανιότερα στη βιβλιογραφία και στην πράξη.
- με ***φυσική γλώσσα*** (natural language) κατά βήματα. Στην περίπτωση αυτή χρειάζεται προσοχή, γιατί μπορεί να παραβιασθεί το τρίτο βασικό χαρακτηριστικό ενός αλγορίθμου, όπως προσδιορίσθηκε προηγουμένως, δηλαδή το κριτήριο του καθορισμού.
- με ***κωδικοποίηση*** (coding), δηλαδή με ένα πρόγραμμα γραμμένο είτε σε μία ψευδογλώσσα είτε σε κάποια γλώσσα προγραμματισμού που όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

Όλοι οι αλγόριθμοι του βιβλίου αυτού είναι κωδικοποιημένοι σε μία υποθετική δομημένη *ψευδογλώσσα*, ωστόσο οι περισσότεροι από αυτούς μπορούν εύκολα σχετικά να προγραμματισθούν σε οποιαδήποτε γλώσσα προγραμματισμού.

2.4 Βασικές συνιστώσες/εντολές ενός αλγορίθμου

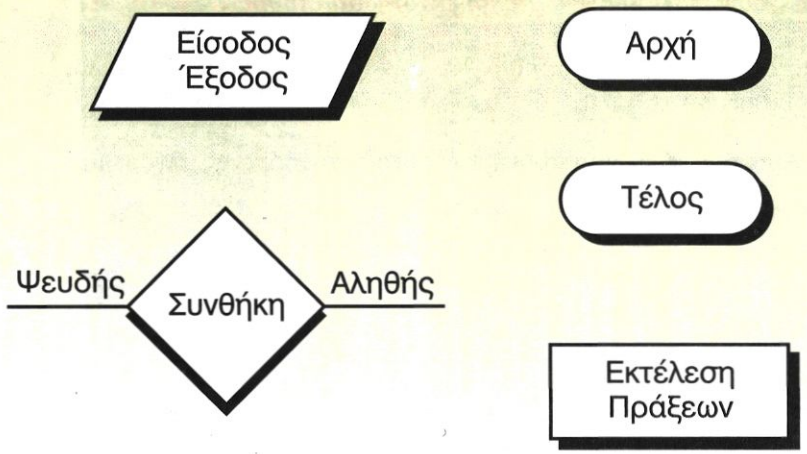
Στη συνέχεια δίνονται παραδείγματα αλγορίθμων όπου εξετάζονται οι

Σύμβολα διαγράμματος ροής

Ένα διάγραμμα ροής αποτελείται από ένα σύνολο γεωμετρικών σχημάτων, όπου το καθένα δηλώνει μία συγκεκριμένη ενέργεια ή λειτουργία. Τα γεωμετρικά σχήματα ενώνονται μεταξύ τους με βέλη, που δηλώνουν τη σειρά εκτέλεσης των ενεργειών αυτών. Τα κυριότερα χρησιμοποιούμενα γεωμετρικά σχήματα είναι τα εξής:

- *έλλειψη*, που δηλώνει την αρχή και το τέλος του κάθε αλγορίθμου,
- *ρόμβος*, που δηλώνει μία ερώτηση με δύο ή περισσότερες εξόδους για απάντηση,
- *ορθογώνιο*, που δηλώνει την εκτέλεση μίας ή περισσότερων πράξεων, και
- *πλάγιο παραλληλόγραμμο*, που δηλώνει είσοδο ή έξοδο στοιχείων. Πολλές φορές το σχήμα αυτό μπορεί να διαφοροποιείται προκειμένου να προσδιορίζεται και το είδος της συσκευής απ' όπου γίνεται η είσοδος ή η έξοδος.

Το επόμενο σχήμα αποτυπώνει όλα αυτά τα σύμβολα..



Μερικά από τα χρησιμοποιούμενα γεωμετρικά σύμβολα στα διαγράμματα ροής.

διάφορες συνιστώσες ενός αλγορίθμου, δηλαδή οι απαραίτητες εντολές ξεκινώντας από τις απλούστερες και προχωρώντας προς τις συνθετότερες. Πιο συγκεκριμένα θα εξετασθούν περιπτώσεις σειριακών εντολών, αναθέσεων τιμών, επιλογής με βάση κριτήρια, διαδικασιών επανάληψης, ενεργειών πολλαπλών επιλογών καθώς και συνδυασμό εμφωλευμένων περιπτώσεων. Για κάθε περίπτωση παρουσιάζονται σχετικά παραδείγματα με την εκφώνηση (σε φυσική γλώσσα), την παρουσίαση των βημάτων που πρέπει να ακολουθηθούν καθώς και το αντίστοιχο διάγραμμα ροής.

2.4.1 Δομή ακολουθίας

Η ακολουθιακή δομή εντολών (σειριακών βημάτων) χρησιμοποιείται πρακτικά για την αντιμετώπιση απλών προβλημάτων, όπου είναι δεδομένη η σειρά εκτέλεσης ενός συνόλου ενεργειών. Ένα απλό παράδειγμα από την καθημερινή ζωή είναι η ακολουθία οδηγιών μίας συνταγής μαγειρικής με στόχο την κατασκευή ενός φαγητού. Τα βήματα και οι ποσότητες που πρέπει να ακολουθηθούν είναι συγκεκριμένα και οι οδηγίες απόλυτα καθορισμένες και σαφείς. Το παράδειγμα που ακολουθεί παρουσιάζει ένα απλό πρόβλημα που επιλύεται με σειριακή εκτέλεση εντολών.

Παράδειγμα 1. ***Ανάγνωση και εκτύπωση αριθμών***

Να διαβασθούν δύο αριθμοί, να υπολογισθεί και να εκτυπωθεί το άθροισμά τους.

Διάβασε=εκτελεστέα εντολή

Αλγόριθμος=δηλωτική εντολή

Από την εκφώνηση προκύπτει αμέσως ο επόμενος αλγόριθμος

```
Αλγόριθμος Παράδειγμα_1
Διάβασε a
Διάβασε b
c ← a + b
Εκτύπωσε c
Τέλος Παράδειγμα_1
```

Ένας αλγόριθμος διατυπωμένος σε ψευδογλώσσα αρχίζει πάντα με τη λέξη *Αλγόριθμος* συνοδευόμενη με το όνομα του αλγορίθμου και τελειώνει με τη λέξη *Τέλος* συνοδευόμενη επίσης με το όνομα του αλγορίθμου. Η πρώτη ενέργεια που γίνεται, είναι η εισαγωγή των δεδομένων. Αυτό επιτυγχάνεται με τη χρήση του ρήματος *Διαβάζω* σε προστακτική. Η λέξη *Διάβασε* συνοδεύεται με το όνομα μίας ή περισσοτέρων μεταβλητών, όπως η a και εννοείται ότι μετά την ολοκλήρωση της ενέργειας αυτή, η μεταβλητή a θα έχει λάβει κάποια αριθμητική τιμή ως περιεχόμενο. Κάθε μία λέξη της

Σταθερές (constants). Με τον όρο αυτό αναφερόμαστε σε προκαθορισμένες τιμές που παραμένουν αμετάβλητες σε όλη τη διάρκεια της εκτέλεσης ενός αλγορίθμου. Οι σταθερές διακρίνονται σε

- αριθμητικές, π.χ. 123, +5, -1,25
- αλφαριθμητικές π.χ. "Τιμή", "Κατάσταση αποτελεσμάτων"
- λογικές που είναι ακριβώς δύο, Αληθής και Ψευδής

Μεταβλητές (variables). Μια μεταβλητή είναι ένα γλωσσικό αντικείμενο, που χρησιμοποιείται για να παραστήσει ένα στοιχείο δεδομένου. Στη μεταβλητή εκχωρείται μια τιμή, η οποία μπορεί να αλλάζει κατά τη διάρκεια εκτέλεσης του αλγορίθμου. Ανάλογα με το είδος της τιμής που μπορούν να λάβουν, οι μεταβλητές διακρίνονται σε αριθμητικές, αλφαριθμητικές και λογικές.

Τελεστές (operators). Πρόκειται για τα γνωστά σύμβολα που χρησιμοποιούνται στις διάφορες πράξεις. Οι τελεστές διακρίνονται σε αριθμητικούς, λογικούς και συγκριτικούς.

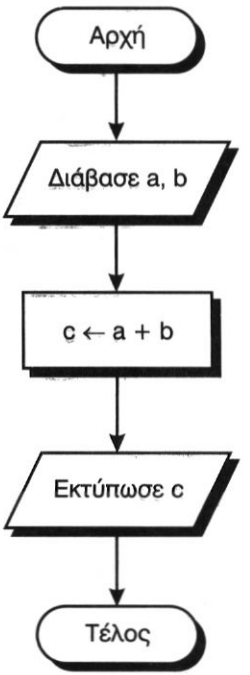
Εκφράσεις (expressions). Οι εκφράσεις διαμορφώνονται από τους τελεστέους (operands), που είναι σταθερές και μεταβλητές και από τους τελεστές. Η διεργασία αποτίμησης μιας έκφρασης συνίσταται στην απόδοση τιμών στις μεταβλητές και στην εκτέλεση των πράξεων. Η τελική τιμή μιας έκφρασης εξαρτάται από την ιεραρχία των πράξεων και τη χρήση των παρενθέσεων. Μια έκφραση μπορεί να αποτελείται από μια μόνο μεταβλητή ή σταθερά μέχρι μια πολύπλοκη μαθηματική παράσταση.

χρησιμοποιούμενης ψευδογλώσσας, που προσδιορίζει μια σαφή ενέργεια, θα αποκαλείται στο εξής *εντολή*. Όλες οι εντολές σε έναν αλγόριθμο αποτυπώνονται με διαφορετικό χρώμα από το όνομα του αλγορίθμου και τις διάφορες σταθερές και μεταβλητές.

Μετά την ανάγνωση των τιμών των μεταβλητών a και b γίνεται ο υπολογισμός του αθροίσματος με την εντολή: c ← a + b. Η εντολή αυτή αποκαλείται εντολή *εκχώρησης τιμής*. Η γενική μορφή της είναι:

Μεταβλητή ← Έκφραση

και η λειτουργία της είναι "γίνονται οι πράξεις στην έκφραση και το αποτέλεσμα αποδίδεται, μεταβιβάζεται, εκχωρείται στη μεταβλητή". Στην εντολή αυτή χρησιμοποιείται το αριστερό βέλος, προκειμένου να δείχνει τη φορά της εκχώρησης. Ας σημειωθεί ότι δεν πρόκειται για εξίσωση, παρ'όλο που σε άλλα βιβλία μπορεί να χρησιμοποιείται το σύμβολο ίσον "=" για τον



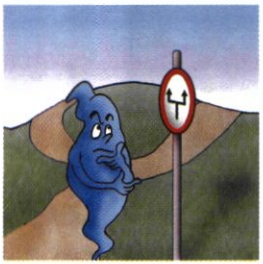
Σχ. 2.1. *Ο αλγόριθμος του παραδείγματος 1 με διάγραμμα ροής*

ίδιο σκοπό. Ας σημειωθεί επίσης ότι οι διάφορες γλώσσες προγραμματισμού χρησιμοποιούν διάφορα σύμβολα για το σκοπό αυτό.

Τέλος ο αλγόριθμος ολοκληρώνεται με την εντολή *Εκτύπωσε*, που αποτυπώνει το τελικό αποτέλεσμα στον εκτυπωτή. Η σύνταξη της εντολής αυτής είναι ανάλογη με αυτή της Διάβασε. Εναλλακτικά μπορεί να χρησιμοποιηθεί και η εντολή *Εμφάνισε*, που αποτυπώνει ένα αποτέλεσμα στην οθόνη.

Στον προηγούμενο αλγόριθμο οι μεταβλητές a και b είναι τα δεδομένα που αποτελούν την *είσοδο*, ενώ η μεταβλητή c αντιπροσωπεύει το αποτέλεσμα, δηλαδή την *έξοδο* του αλγορίθμου. Επιπλέον, ο αλγόριθμος έχει απολύτως καθορισμένη την κάθε εντολή (*καθοριστικότητα*), τελειώνει μετά από συγκεκριμένο αριθμό βημάτων (*περατότητα*), ενώ κάθε εντολή του είναι ιδιαίτερα σαφής και απλή (*αποτελεσματικότητα*). Επομένως ο αλγόριθμος αυτός πληροί τα κριτήρια που χαρακτηρίζουν τον ορισμό της έννοιας του αλγορίθμου, όπως αυτά περιγράφηκαν στην παράγραφο 2.1.

2.4.2 Δομή Επιλογής



Στην πραγματικότητα πολύ λίγα προβλήματα μπορούν να επιλυθούν με τον προηγούμενο τρόπο της σειριακής/ακολουθιακής δομής ενεργειών. Συνήθως τα προβλήματα έχουν κάποιες ιδιαιτερότητες και δεν ισχύουν τα ίδια βήματα για κάθε περίπτωση. Η πλέον συνηθισμένη περίπτωση είναι να λαμβάνονται κάποιες αποφάσεις με βάση κάποια δεδομένα κριτήρια, που μπορεί να είναι διαφορετικά για κάθε διαφορετικό στιγμιότυπο ενός προβλήματος. Οι καθημερινές απλές μας ενέργειες περιέχουν αυτή τη διαδικασία επιλογής με βάση κάποια κατάσταση. Για παράδειγμα, το πρόβλημα της προετοιμασίας μας για έξοδο σχετίζεται με τις καιρικές συνθήκες. Έτσι λέμε ότι, "αν βρέχει, θα πάρω ομπρέλα, αλλιώς θα πάρω καπέλο". Η συνθήκη εδώ είναι το "αν βρέχει", ενώ η απόφαση είναι είτε να πάρω την "ομπρέλα" είτε το "καπέλο" με βάση την "τιμή" της συνθήκης.

Γενικά η διαδικασία της επιλογής περιλαμβάνει τον έλεγχο κάποιας συνθήκης που μπορεί να έχει δύο τιμές (Αληθής ή Ψευδής) και ακολουθεί η απόφαση εκτέλεσης κάποιας ενέργειας με βάση την τιμή της λογικής αυτής συνθήκης. Στη συνέχεια δίνονται δύο παραδείγματα ενεργειών με βάση κάποια συνθήκης επιλογής. Το πρώτο παράδειγμα αφορά στην εκτέλεση κάποιας ενέργειας όταν η συνθήκη είναι Αληθής, ενώ το δεύτερο παράδειγμα αφορά στην εκτέλεση μίας ενέργειας όταν η συνθήκη είναι Αληθής και κάποιας άλλης ενέργειας όταν η συνθήκη είναι Ψευδής.

Παράδειγμα 2. Σύγκριση αριθμών με απλή επιλογή

Να διαβαστεί ένας αριθμός και να εκτυπωθεί η απόλυτη τιμή του.

$|+5| = 5$ *και* $|-5| = 5$

Όπως είναι γνωστό, η απόλυτη τιμή ενός αριθμού είναι ο ίδιος ο αριθμός, αν αυτός είναι θετικός ή μηδέν και ο αντίθετός του, αν είναι αρνητικός. Έτσι προκειμένου να βρεθεί η απόλυτη τιμή, αρκεί να ελεγχθεί, αν τυχόν ο δεδομένος αριθμός είναι αρνητικός, οπότε στην περίπτωση αυτή πρέπει να βρεθεί ο αντίθετός του. Ο συλλογισμός αυτός οδηγεί στον επόμενο αλγόριθμο.

```
Αλγόριθμος Παράδειγμα_2
Διάβασε a
Αν a < 0 τότε a ← a*(-1)
Εκτύπωσε a
Τέλος Παράδειγμα_2
```

Στην παράσταση αλγορίθμων με ψευδογλώσσα η επιλογή υλοποιείται με την εντολή *Αν...τότε*. Η σύνταξη της εντολής είναι:

Αν *συνθήκη* **τότε** *εντολή*

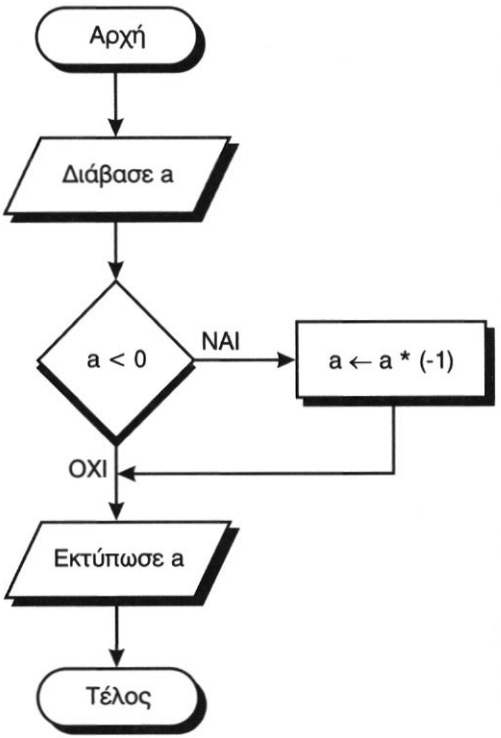
και η λειτουργία της είναι: Αν ισχύει η συνθήκη (δηλαδή αν είναι αληθής), τότε μόνο εκτελείται η εντολή. Σε κάθε περίπτωση εκτελείται στη συνέχεια η εντολή, που ακολουθεί.

Η συνθήκη είναι μια λογική έκφραση.

Στην εντολή *Αν...τότε* είναι πιθανό, όταν ισχύει η συνθήκη, να απαιτείται η εκτέλεση περισσότερων από μία εντολές. Στην περίπτωση αυτή οι διαδοχικές εντολές γράφονται από κάτω και σε εσοχή, ενώ το σχήμα επιλογής κλείνει με τη λέξη Τέλος_αν. Π.χ

```
Αν συνθήκη τότε
    εντολή_1
    εντολή_2
    ...............
    εντολή_ν
Τέλος_αν
```

Όπως και στον αλγόριθμο του προηγούμενου παραδείγματος, εύκολα προκύπτει ότι η τιμή a είναι και *είσοδος* αλλά και *έξοδος* του αλγορίθμου. Επιπλέον, ο αλγόριθμος έχει καθορισμένη κάθε του εντολή (*καθοριστικότητα*), τελειώνει μετά από πεπερασμένο αριθμό βημάτων (*περατότητα*), ενώ κάθε εντολή του είναι ιδιαίτερα απλή κατά την εκτέλεσή της (*αποτελεσματικότητα*). Έτσι προκύπτει ότι ο αλγόριθμος αυτός πράγματι πληροί τα κριτήρια που περιγράφηκαν στην παράγραφο 2.1.



Σχ. 2.2. Ο αλγόριθμος του παραδείγματος 2 με διάγραμμα ροής

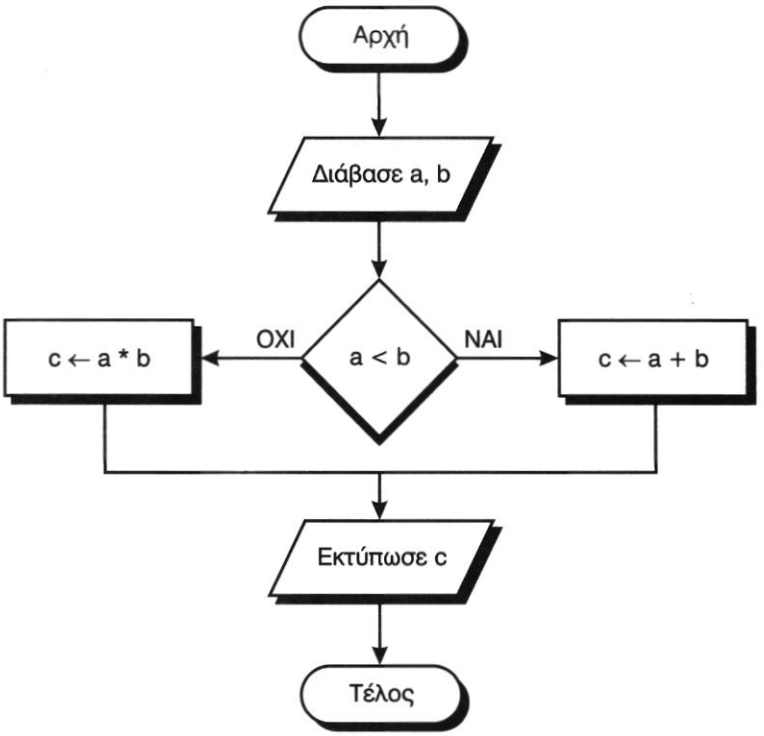
Παράδειγμα 3. Σύγκριση αριθμών με σύνθετη επιλογή

Να διαβασθούν δύο αριθμοί και σε περίπτωση που ο πρώτος αριθμός είναι μικρότερος του δεύτερου, να υπολογισθεί και να εκτυπωθεί το άθροισμά τους, διαφορετικά να υπολογισθεί και να εκτυπωθεί το γινόμενό τους.

```
Αλγόριθμος Παράδειγμα_3
Διάβασε a, b
Αν a < b τότε
    c ← a + b
αλλιώς
    c ← a * b
Τέλος_αν
Εκτύπωσε c
Τέλος Παράδειγμα_3
```

Στο παράδειγμα αυτό χρησιμοποιείται η γενική μορφή της εντολής επιλογής, που είναι:

```
Αν συνθήκη τότε
    εντολή ή εντολές
αλλιώς
    εντολή ή εντολές
Τέλος_αν
```



Σχ. 2.3. Ο αλγόριθμος του παραδείγματος 3 με διάγραμμα ροής

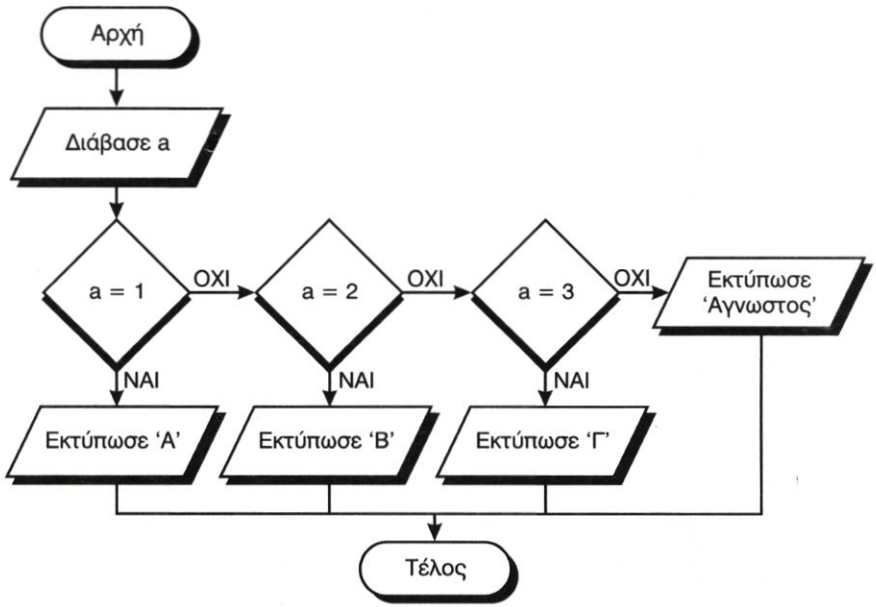
2.4.3 Διαδικασίες πολλαπλών επιλογών

Οι διαδικασίες των πολλαπλών επιλογών εφαρμόζονται στα προβλήματα όπου μπορεί να ληφθούν διαφορετικές αποφάσεις ανάλογα με την τιμή που παίρνει μία έκφραση. Για παράδειγμα, κάθε γράμμα της αλφαβήτου μπορεί να αντιστοιχιθεί σε κάποιον ακέραιο αριθμό από το 1 μέχρι και 24, για τις ανάγκες κάποιας κωδικοποίησης. Στο παράδειγμα που ακολουθεί παρουσιάζεται μία περίπτωση πολλαπλών επιλογών με διαφορετική ακολουθία εντολών σε κάθε περίπτωση.

Παράδειγμα 4. Ανάθεση γραμμάτων σε αριθμούς

Να διαβασθεί ένας ακέραιος και να εκτυπωθεί το αντίστοιχο γράμμα της αλφαβήτου, αν ο ακέραιος έχει τιμή 1 ή 2 ή 3 διαφορετικά να εκτυπωθεί η λέξη "άγνωστος".

```
Αλγόριθμος Παράδειγμα_4
Διάβασε a
Αν a = 1 τότε εκτύπωσε "Α"
αλλιώς_αν a = 2 τότε εκτύπωσε "Β"
αλλιώς_αν a = 3 τότε εκτύπωσε "Γ"
αλλιώς εκτύπωσε "άγνωστος"
Τέλος_αν
Τέλος Παράδειγμα_4
```



Σχ. 2.4. *Ο αλγόριθμος του παραδείγματος 4 με διάγραμμα ροής*

Αν οι διαφορετικές επιλογές είναι πολλές, τότε είναι προτιμότερο να χρησιμοποιηθεί το σχήμα πολλαπλής επιλογής *Επίλεξε...Τέλος επιλογών* (select case), όπως στο παράδειγμα που ακολουθεί.

Παράδειγμα 5. ***Επιλογή ορίων***

Να εισαχθεί ένας ακέραιος που αντιστοιχεί σε μια ηλικία και να βρεθεί σε ποια όρια εντάσσεται η δεδομένη ηλικία εμφανίζοντας σχετικό μήνυμα.

```
Αλγόριθμος Παράδειγμα 5.
Εμφάνισε "Σε ποια ηλικία άρχισες να μαθαίνεις προγραμ-
ματισμό;"
Διάβασε age
Επίλεξε age
    Περίπτωση < 0
        Εμφάνισε "Είπαμε ηλικία ..."
    Περίπτωση < 5
        Εμφάνισε "Μάλλον τα παραλές !!"
    Περίπτωση < 60
        Εμφάνισε "Μπράβο"
    Περίπτωση < 100
        Εμφάνισε "Ποτέ δεν είναι αργά"
    Περίπτωση αλλιώς
        Εμφάνισε "Κάλλιο αργά παρά ποτέ"
Τέλος_επιλογών
Τέλος Παράδειγμα_5
```

Στο σχήμα πολλαπλής επιλογής Επίλεξε... Τέλος_επιλογών, που χρησιμοποιήθηκε στο παράδειγμα αυτό, εξετάζεται μια έκφραση (εδώ είναι μια μόνο μεταβλητή, η age). Ανάλογα με την τιμή της έκφρασης εκτελούνται οι εντολές μετά την Περίπτωση που αντιστοιχεί στην τιμή της έκφρασης. Οι τιμές που συνοδεύουν κάθε Περίπτωση μπορεί να είναι μία ή περισσότερες διάκριτες τιμές, περιοχή τιμών από - έως ή να υπακούουν σε μια συνθήκη (όπως στο παράδειγμα αυτό). Αν η τιμή της έκφρασης δεν αντιστοιχεί σε καμία Περίπτωση, τότε εκτελούνται οι εντολές που ακολουθούν την Περίπτωση αλλιώς. Μετά την εκτέλεση μιας περίπτωσης, ο αλγόριθμος συνεχίζει με την εντολή, που ακολουθεί το Τέλος_επιλογών

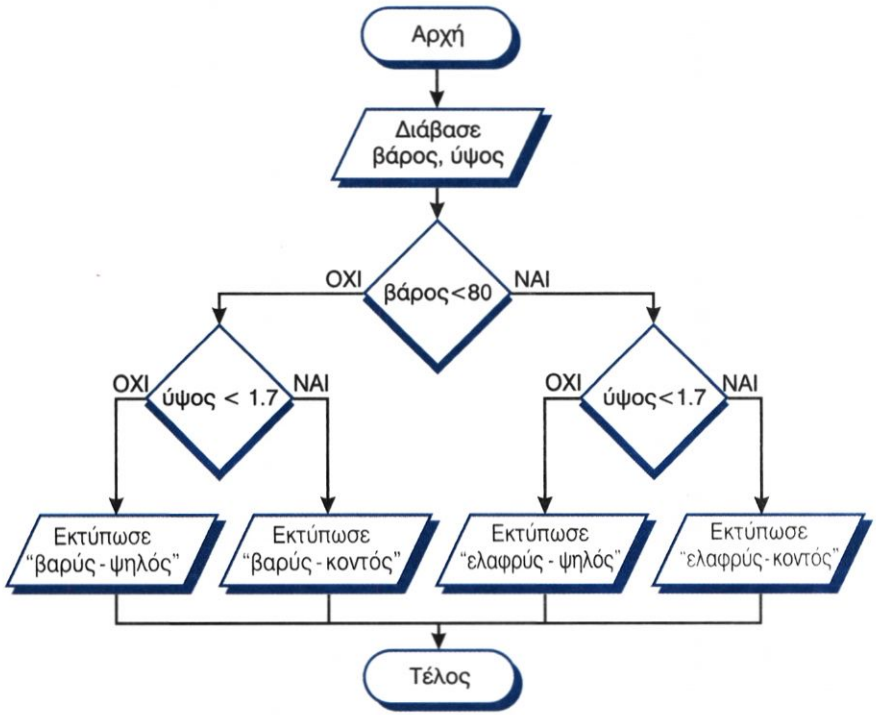
2.4.4 Εμφωλευμένες Διαδικασίες

Πολλαπλές επιλογές μπορούν να γίνουν και με μία εμφωλευμένη δομή. Το επόμενο παράδειγμα περιγράφει τον τρόπο με τον οποίο μία εντολή *Αν...τότε* είναι η εντολή που εκτελείται, όταν ισχύει (ή δεν ισχύει) η συνθήκη μίας άλλης εντολής *Αν...τότε*. Βέβαια η λογική αυτή μπορεί να επεκταθεί, δηλαδή να έχουμε νέα εμφωλευμένη δομή μέσα σε μία εμφωλευμένη δομή κοκ.

Παράδειγμα 6. ***Χαρακτηρισμός ατόμων***

Να διαβάζονται δύο αριθμοί που αντιστοιχούν στο ύψος και βάρος ενός άνδρα. Να εκτυπώνεται ότι ο άνδρας είναι "ελαφρύς", αν το βάρος του είναι κάτω από 80 κιλά, ή να εκτυπώνεται "βαρύς" στην αντίθετη περίπτωση. Επίσης να εκτυπώνεται "κοντός" αν το ύψος του είναι κάτω από 1.70, αλλιώς να εκτυπώνεται "ψηλός".

```
Αλγόριθμος Παράδειγμα_6
Διάβασε βάρος, ύψος
Αν βάρος < 80 τότε
    Αν ύψος < 1.70  τότε
        εκτύπωσε "ελαφρύς-κοντός"
    αλλιώς
        εκτύπωσε "ελαφρύς-ψηλός"
    Τέλος_αν
αλλιώς
    Αν ύψος < 1.70 τότε
        εκτύπωσε "βαρύς-κοντός"
    αλλιώς
        εκτύπωσε "βαρύς-ψηλός"
    Τέλος_αν
Τέλος_αν
Τέλος Παράδειγμα_6
```



Σχ. 2.5. *Ο αλγόριθμος του παραδείγματος 6 με διάγραμμα ροής*

Σε πολλές περιπτώσεις η συνθήκη είναι αρκετά πιο "δύσκολη", δηλαδή

εμπεριέχει αποφάσεις που πιθανόν να βασίζονται σε περισσότερα από ένα κριτήρια. Ο συνδυασμός των κριτηρίων αυτών καθορίζει και τις "λογικές" πράξεις που μπορούν να γίνουν μεταξύ διαφορετικών συνθηκών. Πολύ συχνά στην καθημερινή ζωή κάποιες αποφάσεις βασίζονται σε συνδυασμούς κριτηρίων και λογικών πράξεων. Για παράδειγμα, το πρόβλημα της προετοιμασίας μας για έξοδο μπορεί να επεκταθεί ως εξής "αν βρέχει **ή** αν χιονίζει θα πάρω ομπρέλα", είτε στην πρόταση "αν έχει ήλιο **και** αν έχει ζέστη θα πάρω καπέλο", είτε στην πρόταση "αν **δεν** έχει ήλιο θα πάρω ομπρέλα". Οι τρεις αυτές προτάσεις περιγράφουν και τις τρεις λογικές πράξεις που μπορεί να ισχύουν μεταξύ διαφορετικών συνθηκών. Η λογική πράξη **ή** είναι αληθής όταν οποιαδήποτε από τις δύο προτάσεις είναι αληθής. Η λογική πράξη **και** είναι αληθής όταν και οι δύο προτάσεις είναι αληθείς ενώ η λογική πράξη **όχι** (η λέξη "δεν" στο παράδειγμά μας) είναι αληθής όταν η πρόταση που την ακολουθεί είναι ψευδής. Ο επόμενος πίνακας δίνει τις τιμές των τριών αυτών λογικών πράξεων για όλους τους συνδυασμούς τιμών.

Πρόταση Α	Πρόταση Β	Α ή Β	Α και Β	όχι Α
Αληθής	Αληθής	Αληθής	Αληθής	Ψευδής
Αληθής	Ψευδής	Αληθής	Ψευδής	Ψευδής
Ψευδής	Αληθής	Αληθής	Ψευδής	Αληθής
Ψευδής	Ψευδής	Ψευδής	Ψευδής	Αληθής

2.4.5 Δομή Επανάληψης

Η διαδικασία της επανάληψης είναι ιδιαίτερα συχνή, αφού πλήθος προβλημάτων μπορούν να επιλυθούν με κατάλληλες επαναληπτικές διαδικασίες. Η λογική των επαναληπτικών διαδικασιών εφαρμόζεται στις περιπτώσεις, όπου μία ακολουθία εντολών πρέπει να εφαρμοσθεί σε ένα σύνολο περιπτώσεων, που έχουν κάτι κοινό. Για παράδειγμα, όλες οι τράπεζες κάθε εξάμηνο αποδίδουν τόκους των καταθέσεων ταμιευτηρίου. Ο υπολογισμός των τόκων πρέπει να γίνει για όλους τους λογαριασμούς της τράπεζας, άρα η πράξη

τόκος = ποσό * επιτόκιο

πρέπει να εκτελεσθεί για όλους τους τραπεζικούς λογαριασμούς. Οι επαναληπτικές διαδικασίες μπορεί να έχουν διάφορες μορφές και συνήθως εμπεριέχουν και συνθήκες επιλογών (όπως αυτές περιγράφηκαν στην προηγούμενη υποπαράγραφο). Στη συνέχεια δίνεται ένα σύνολο παρα-

δειγμάτων που εντάσσονται στις πλέον γνωστές κατηγορίες επαναληπτικών διαδικασιών.

Παράδειγμα 7. Εκτύπωση διαδοχικών αριθμών με επαναληπτική εντολή: όσο...επανάλαβε

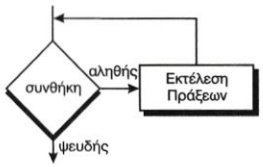
Να γραφεί αλγόριθμος που να εμφανίζει τους αριθμούς από 1 έως 100.

i ← i + 1 ⇒ "η νέα τιμή της μεταβλητής i είναι η προηγούμενη συν ένα"

Στον αλγόριθμο αυτό επιζητείται η παρουσίαση μίας σειράς αριθμών. Αν οι αριθμοί αυτοί ήσαν λίγοι, τότε αυτό θα μπορούσε να γίνει με την παράθεση αντίστοιχων εντολών εμφάνισης. Το ίδιο θα συμβεί και στην περίπτωση που οι αριθμοί είναι περισσότεροι, αλλά δεν έχουν καμία σχέση μεταξύ τους π.χ. 5, 207, -32 κοκ. Όμως στο ζητούμενο αλγόριθμο παρατηρούμε ότι κάθε αριθμός παράγεται από τον προηγούμενό του με απλό τρόπο δηλαδή προσθέτοντας κάθε φορά το 1. Μπορεί λοιπόν να χρησιμοποιηθεί μια μεταβλητή, έστω i, η οποία αρχίζει από το 1 και καταλήγει στο 100 αυξανόμενη κατά 1. Η εκάστοτε αύξηση της μεταβλητής αυτής μπορεί να γίνει με τη χρήση της εντολής εκχώρησης

i ← i + 1

Η αρχική τιμή της μεταβλητής i ορίζεται εύκολα με την εντολή i ← 1. Το ζητούμενο είναι να εκτελεστεί 100 φορές η εντολή i ← i + 1. Αυτό επιτυγχάνεται με τη χρήση της εντολής *Όσο ...επανάλαβε*. Η σύνταξη της εντολής αυτής είναι:




```
Όσο συνθήκη επανάλαβε
    εντολές
Τέλος_επανάληψης
```

Η λειτουργία της εντολής είναι η εξής: Επαναλαμβάνεται η εκτέλεση των εντολών, όσο η συνθήκη είναι αληθής. Όταν η συνθήκη γίνει ψευδής, τότε ο αλγόριθμος συνεχίζεται με την εντολή που ακολουθεί το *'Τέλος_επανάληψης'*. Με την εισαγωγή της εντολής αυτής η σχεδίαση του ζητούμενου αλγορίθμου είναι:

*Το τμήμα του αλγόριθμου που επαναλαμβάνεται, δηλαδή από την εντολή Όσο μέχρι το Τέλος_επανάληψης αποκαλείται **βρόχος**.*

```
Αλγόριθμος Παράδειγμα_7
i ← 1
Όσο i ≤ 100 επανάλαβε
    Εμφάνισε i
    i ← i + 1
Τέλος_επανάληψης
Τέλος Παράδειγμα_7
```

Παράδειγμα 8: **Επαναληπτική είσοδος στοιχείων**

Να γραφεί αλγόριθμος που να διαβάζει ένα άγνωστο πλήθος αριθμών και να εμφανίζει τον κάθε αριθμό.

Το πρόβλημα αυτό παρουσιάζει την εξής ιδιομορφία: ενώ φαίνεται ότι θα χρησιμοποιηθεί για τη λύση του κάποια επαναληπτική διαδικασία, δεν προσδιορίζεται ο τρόπος τερματισμού της. Κατ'αρχήν, λοιπόν, ας εξετάσουμε τον αλγόριθμο που εκτελεί ένας άνθρωπος, όταν αντιγράφει κάποιους αριθμούς, όπως για παράδειγμα όταν συγκεντρώνονται τα έξοδα από διάφορους λογαριασμούς. Ο αλγόριθμος αυτός είναι:

Βήμα 1. Διάβασε έναν αριθμό

Βήμα 2. Γράψε τον αριθμό

Βήμα 3. Επανάλαβε τη διαδικασία από το βήμα 1.

Ο αλγόριθμος αυτός έχει μια ατέλεια, δεν διαθέτει τρόπο τερματισμού (ατέρμων βρόχος). Η έλλειψη αυτή είναι φυσική, εφ'όσον ο αλγόριθμος εκτελείται από έναν άνθρωπο. Αυτός θα σταματήσει να γράφει, όταν δεν υπάρχουν πλέον άλλοι αριθμοί. Ωστόσο είναι δυνατόν να διορθωθεί αυτή η ατέλεια, αν το βήμα 3 λάβει την εξής μορφή:

Βήμα 3. Αν υπάρχουν άλλοι αριθμοί, επανάλαβε τη διαδικασία από το βήμα 1, αλλιώς σταμάτησε.

Ο αλγόριθμος τώρα είναι σωστός και μπορεί να εκτελεστεί και από μία μηχανή. Όμως έχει ένα άλλο μειονέκτημα: ο τερματισμός γίνεται μέσα από την εντολή Αν ... τότε ... αλλιώς ..., πράγμα που δεν συνιστάται και πρέπει να αποφεύγεται, γιατί εύκολα μπορεί να χάσει ο προγραμματιστής τον έλεγχο της ροής του προγράμματος και να οδηγηθεί σε λάθος. Για την άρση του μειονεκτήματος αυτού πρέπει να χρησιμοποιηθεί μία εντολή επαναληπτικής διαδικασίας, όπως η εντολή *Όσο...επανάλαβε*. Ο τελικός αλγόριθμος είναι ο εξής:

Ο βρόχος επανάληψης μπορεί να μην εκτελεσθεί καμία φορά, αν η πρώτη τιμή που διαβάζεται είναι αρνητική.

```
Αλγόριθμος Παράδειγμα_8
Διάβασε x
Όσο x > 0 επανάλαβε
    Εμφάνισε x
    Διάβασε x
Τέλος_επανάληψης
Τέλος Παράδειγμα_8
```

Στον προηγούμενο αλγόριθμο η επαναληπτική διαδικασία τερματίζεται, όταν διαβασθεί ένας αρνητικός ή μηδενικός αριθμός. Δηλαδή, θεωρείται ότι

οι εισαγόμενοι αριθμοί πρέπει να είναι θετικοί. Αν αυτό δεν συμβαίνει, τότε μπορεί να χρησιμοποιηθεί ως συνθήκη τερματισμού οποιαδήποτε συγκεκριμένη τιμή έχει συμφωνηθεί, ότι θα χρησιμοποιείται για το σκοπό αυτό, π.χ. η 999999. Προφανώς αυτή η τιμή δεν μπορεί να ανήκει στις εισαγόμενες τιμές. Στην περίπτωση αυτή η εντολή *Όσο...επανάλαβε* θα γραφεί ως εξής:

Όσο x ≠ 999999 **επανάλαβε**

Παράδειγμα 9. Εκτύπωση θετικών αριθμών με εντολή: αρχή_επανάληψης...μέχρις_ότου

Να διαβάζονται και να εκτυπώνονται όσοι θετικοί αριθμοί δίνονται από το πληκτρολόγιο. Ο αλγόριθμος τελειώνει, όταν δοθεί ένας αρνητικός αριθμός.

```
Αλγόριθμος Παράδειγμα_9
Αρχή_επανάληψης
    Διάβασε x
    Εμφάνισε x
Μέχρις_ότου x < 0
Τέλος Παράδειγμα_9
```

Η εντολή Αρχή_επανάληψης...Μέχρις_ότου εκτελείται οπωσδήποτε μια φορά

Ας σημειωθεί ότι, στο παράδειγμα αυτό ο βρόχος επανάληψης θα εκτελεσθεί οπωσδήποτε τουλάχιστον μία φορά ακόμα και αν η αρχική τιμή της μεταβλητής x είναι αρνητική. Η βασική διαφοροποίηση αυτής της μορφής επαναληπτικής διαδικασίας σε σχέση με την επαναληπτική διαδικασία που παρουσιάσθηκε στο προηγούμενο παράδειγμα, οφείλεται στη θέση της λογικής συνθήκης στη ροή εκτέλεσης των εντολών.

Παράδειγμα 10. Υπολογισμός αθροίσματος αριθμών με επαναληπτική εντολή: για...από...μέχρι

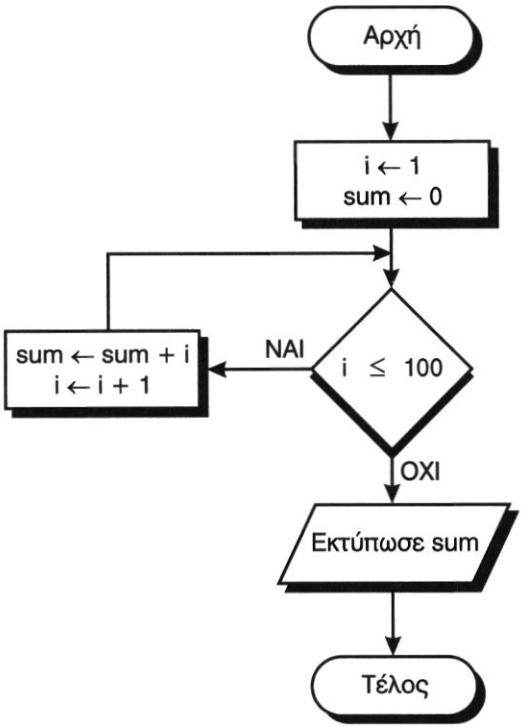
Να βρεθεί και να εκτυπωθεί το άθροισμα των 100 ακεραίων από το 1 μέχρι το 100.

Οταν ο αριθμός των φορών που θα εκτελεστεί μια επαναληπτική διαδικασία είναι γνωστός εκ των προτέρων, τότε είναι προτιμότερο να χρησιμοποιείται η εντολή Για...από...μέχρι. Ετσι ο ζητούμενος αλγόριθμος είναι.

```
Αλγόριθμος Παράδειγμα_10
Sum ← 0
Για i από 1 μέχρι 100
    Sum ← Sum + i
Τέλος_επανάληψης
Εκτύπωσε Sum
Τέλος Παράδειγμα_10
```

Sum ← Sum + i ⇒ Η νέα τιμή του Sum είναι η παλιά συν i

Οπως γίνεται φανερό, η εντολή Για...από...μέχρι περιλαμβάνει όλα τα απαιτούμενα στοιχεία για την επανάληψη, δηλαδή αρχική τιμή της μεταβλητής i (=1) και τελική τιμή (=100). Το βήμα μεταβολής της μεταβλητής i είναι 1, το οποίο υπονοείται και δεν σημειώνεται, όταν είναι 1. Η μεταβλητή Sum που υποδέχεται το άθροισμα των διαδοχικών αριθμών, πρέπει να εκκινήσει με τιμή 0, ενώ το εκάστοτε μερικό άθροισμα υπολογίζεται με την εντολή εκχώρησης εντός του βρόχου. Στο τέλος η μεταβλητή Sum θα περιέχει το τελικό άθροισμα.



Σχ. 2.6. *Ο αλγόριθμος του παραδείγματος 10 με διάγραμμα ροής*

Παράδειγμα 11. ***Υπολογισμός αθροίσματος με επαναληπτική εντολή: για...από...μέχρι...με_βήμα***

Να βρεθεί και να εκτυπωθεί το άθροισμα των άρτιων αριθμών από το 1 μέχρι το 100.

Η λύση αυτού του προβλήματος είναι παρόμοια με αυτή του προηγούμενου. Η μόνη αλλαγή είναι στην εντολή επανάληψης όπου προσδιορίζεται η ποσότητα *βήμα* , η οποία κάθε φορά προστίθεται στην τιμή της μεταβλητής i. Έτσι έχουμε

```
Αλγόριθμος Παράδειγμα_11
άθροισμα ← 0
Για i από 2 μέχρι 100 με_βήμα 2
    άθροισμα ← άθροισμα + i
Τέλος_επανάληψης
Εκτύπωσε άθροισμα
Τέλος Παράδειγμα_11
```

Από τα προηγούμενα δύο παραδείγματα γίνεται φανερός ο τρόπος χρήσης της εντολής Για...από...μέχρι. Ας σημειωθεί ωστόσο, ότι υπάρχουν κάποιες δεσμεύσεις μεταξύ των τιμών από, μέχρι και βήμα. Ετσι το βήμα δεν μπορεί να είναι μηδέν, γιατί τότε ο βρόχος εκτελείται επ' άπειρον. Είναι δυνατόν όμως το βήμα να έχει αρνητική τιμή, αρκεί η τιμή από να είναι μεγαλύτερη από την τιμή μέχρι, όπως για παράδειγμα στην επόμενη εντολή:

*Ο βρόχος **Για** k **από** 5 **μέχρι** 5 εκτελείται ακριβώς μία φορά*

*Ο βρόχος **Για** k **από** 5 **μέχρι** 1 δεν εκτελείται καμία φορά*

Για k **από** 100 **μέχρι** 0 **με_βήμα** -1

Επίσης οι τιμές από, μέχρι και βήμα δεν είναι απαραίτητο να είναι ακέραιες. Μπορούν λάβουν οποιαδήποτε πραγματική τιμή. Για παράδειγμα, όταν ζητείται να βρεθούν διαδοχικές τιμές μιας συνάρτησης f(x) για x από 0 έως 1, τότε μπορεί να γραφεί η επόμενη εντολή

Για x **από** 0 **μέχρι** 1 **με_βήμα** 0,01

Παράδειγμα 12. ***Πολλαπλασιασμός αλά ρωσικά***

Στη συνέχεια προχωρούμε στην ανάπτυξη ενός συνθετότερου προβλήματος, όπου για την αλγοριθμική του επίλυση γίνεται χρήση αρκετών από τις προηγούμενες δομές. Ας θεωρήσουμε την πράξη του πολλαπλασιασμού δύο ακεραίων αριθμών και ας θυμηθούμε πως αυτή υλοποιείται χειρωνακτικά. Τοποθετούμε, λοιπόν, τους δύο αριθμούς τον ένα κάτω από τον άλλο και πολλαπλασιάζουμε κάθε ψηφίο του κάτω αριθμού με όλα ψηφία του επάνω αριθμού. Πιο συγκεκριμένα, για κάθε ψηφίο του κάτω αριθμού παράγεται ένα μερικό γινόμενο, ενώ τα μερικά γινόμενα τοποθετούνται το

ένα κάτω από το άλλο με μία μετατόπιση από τα δεξιά προς τα αριστερά καθώς θεωρούμε διαδοχικά τα ψηφία των μονάδων, των δεκάδων, των εκατοντάδων κ.λπ. Στη συνέχεια γίνεται η πρόσθεση των επιμέρους γινομένων, αφού τα τοποθετήσουμε στην κατάλληλη διάταξη όπως φαίνεται στο σχήμα 2.7.

```
    45
x   19
  ----
   405
+ 45
  ----
   855
```

Σχ. 2.7. *Χειρωνακτικός τρόπος πολλαπλασιασμού.*

Ωστόσο, η πράξη του πολλαπλασιασμού δεν εκτελείται από τον υπολογιστή με τον τρόπο αυτό. Πιο συγκεκριμένα, ο χρησιμοποιούμενος τρόπος είναι ο λεγόμενος *πολλαπλασιασμός αλά ρωσικά*. Χωρίς βλάβη της γενικότητας θεωρούμε ότι οι ακέραιοι είναι θετικοί (μεγαλύτεροι του μηδενός), αλλά η μέθοδος μπορεί εύκολα να μετατραπεί, ώστε να περιγράψει και την περίπτωση των αρνητικών ακεραίων. Πώς ακριβώς λειτουργεί η μέθοδος, θα φανεί με το επόμενο παράδειγμα, όπου περιγράφεται ο αλγόριθμος με ελεύθερο κείμενο.

Έστω, λοιπόν, ότι δίνονται δύο θετικοί ακέραιοι αριθμοί, οι αριθμοί 45 και 19. Οι αριθμοί γράφονται δίπλα-δίπλα και ο πρώτος διπλασιάζεται, ενώ ο δεύτερος υποδιπλασιάζεται αγνοώντας το δεκαδικό μέρος. Στο σχήμα 2.8 παρουσιάζεται η επαναλαμβανόμενη διαδικασία, που συνεχίζεται μέχρις ότου στη δεύτερη στήλη να προκύψει μονάδα. Τελικώς, το γινόμενο ισούται με το άρθοισμα των στοιχείων της πρώτης στήλης, όπου αντίστοιχα στη δεύτερη στήλη υπάρχει περιττός αριθμός. Για το παράδειγμά μας, τα στοιχεία αυτά παρουσιάζονται στην τρίτη στήλη.

45	19	45
90	9	90
180	4	
360	2	
720	1	720
Άθροισμα =		855

Σχ. 2.8. *Πολλαπλασιασμός αλά ρωσικά.*

Ολίσθηση (shift)

Στα κυκλώματα του υπολογιστή τα δεδομένα αποθηκεύονται με δυαδική μορφή, δηλαδή 0 και 1, ανεξάρτητα από το πως τα ορίζει ο προγραμματιστής, όπως ακεραίους ή πραγματικούς σε δεκαδικό σύστημα, ή ακόμη χαρακτήρες κ.λπ. Έτσι ο αριθμός 17 του δεκαδικού συστήματος ισοδυναμεί με τον αριθμό 00010001 του δυαδικού συστήματος, ο οποίος μπορεί να αποθηκευθεί σε ένα byte. Αν μετακινήσουμε τα ψηφία αυτά κατά μία θέση προς τα αριστερά, δηλαδή αν προσθέσουμε ένα 0 στο τέλος του αριθμού και αγνοήσουμε το αρχικό 0, τότε προκύπτει ο αριθμός 00100010 του δυαδικού συστήματος, που ισοδυναμεί με το αριθμό 34 του δεκαδικού συστήματος. Επίσης, με παρόμοιο τρόπο, αν μετακινήσουμε τα ψηφία κατά μία θέση δεξιά, δηλαδή αποκόψουμε το τελευταίο ψηφίο 1 και θεωρήσουμε ένα ακόμη αρχικό 0, τότε προκύπτει ο αριθμός 00001000 του δυαδικού συστήματος, που ισοδυναμεί με τον αριθμό 8 του δεκαδικού συστήματος. Άρα η ολίσθηση προς τα αριστερά ισοδυναμεί με πολλαπλασιασμό επί δύο, ενώ η ολίσθηση προς τα δεξιά ισοδυναμεί με την ακέραια διαίρεση διά δύο.

Στοιχεία ψευδογλώσσας

1. Σταθερές

Αριθμητικές: χρησιμοποιούνται οι αριθμητικοί χαρακτήρες, το +, το – και το κόμμα ως δεκαδικό σημείο,
Αλφαριθμητικές: σχηματίζονται από οποιουσδήποτε χαρακτήρες εντός διπλών εισαγωγικών,
Λογικές: υπάρχουν δύο, οι Αληθής και Ψευδής.

2. Μεταβλητές

Για τη σύνθεση του ονόματος μιας μεταβλητής χρησιμοποιούνται οι αριθμητικοί χαρακτήρες, οι αλφαβητικοί χαρακτήρες πεζοί και κεφαλαίοι, καθώς και ο χαρακτήρας _ (underscore). Οι μεταβλητές μπορούν επίσης να είναι αριθμητικές, αλφαριθμητικές και λογικές.

3. Τελεστές

Αριθμητικοί: **+**, **–**, *****, **/**, **^**, **div**, **mod**
Συγκριτικοί: **≤**, **<**, **=**, **≠**, **>**, **≥**
Λογικοί: **και** (σύζευξη), **ή** (διάζευξη), **όχι** (άρνηση).

4. Εκφράσεις

Σχηματίζονται από σταθερές, μεταβλητές, συναρτήσεις, τελεστές και παρενθέσεις.

5. Εντολή εκχώρησης

```
Μεταβλητή ← έκφραση
```

6. Σχήματα λογικών υποθέσεων

```
Αν <συνθήκη> τότε <εντολή>
```

```
Αν <συνθήκη> τότε
    <διαδικασία_1>
αλλιώς
    <διαδικασία_2>
Τέλος_αν
```

```
Επίλεξε έκφραση
    Περίπτωση 1
        Διαδικασία_1
    ............
    Περίπτωση ν
        Διαδικασία _ν
    Περίπτωση αλλιώς
        Διαδικασία _αλλιώς
Τέλος_επιλογών
```

```
Αν <συνθήκη_1> τότε
    <διαδικασία_1>
αλλιώς_αν <συνθήκη_2> τότε
    <διαδικασία_2>
......
αλλιώς_αν <συνθήκη_ν> τότε
    <διαδικασία_ν>
αλλιώς
    <διαδικασία_αλλιώς>
Τέλος_αν
```

όπου ως διαδικασία λαμβάνεται ένα σύνολο εντολών

7. Επαναληπτικές διαδικασίες

⇨ Επαναληπτικό σχήμα με έλεγχο επανάληψης στην αρχή

```
Όσο <συνθήκη> επανάλαβε
    Διαδικασία
Τέλος_επανάληψης
```

⇨ Επαναληπτικό σχήμα με έλεγχο επανάληψης στο τέλος

```
Αρχή_επανάληψης
    Διαδικασία
Μέχρις_ότου <συνθήκη>
```

⇨ Επαναληπτικό σχήμα ορισμένων φορών επανάληψης

```
Για μεταβλητή από τ1 μέχρι τ2 με_βήμα β
    Διαδικασία
Τέλος_επανάληψης
```

8. Ρήματα σε προστακτική

Για παράδειγμα, "Διάβασε", "Γράψε", "Εκτέλεσε" κ.λπ.

9. Ουσιαστικά

Σε ορισμένες περιπτώσεις όταν οι ζητούμενες ενέργειες είναι πολλές ή προφανείς, καθορίζονται με τη χρήση ουσιαστικών αντί ρημάτων, όπως "εισαγωγή δεδομένων", "εμφάνιση πεδίων στην οθόνη" κ.λπ.

10. Σχόλια

Προκειμένου να διαχωρίζονται οι επεξηγηματικές φράσεις από τις λέξεις-κλειδιά του αλγορίθμου, στις πρώτες προτάσσεται το σύμβολο !, για παράδειγμα !Σχόλια.

11. Πρώτη και τελευταία γραμμή ενός αλγορίθμου είναι αντίστοιχα **Αλγόριθμος** <όνομα_αλγορίθμου> και **Τέλος** <όνομα_αλγορίθμου>

12. Δεδομένα και αποτελέσματα

Τα δεδομένα εισόδου (αν υπάρχουν) περιγράφονται στη δεύτερη γραμμή του αλγορίθμου εντός των συμβόλων // ... //. Αντίστοιχα τα αποτελέσματα εξόδου δίνονται στην προτελευταία γραμμή του αλγορίθμου εντός των συμβόλων // ... //.

Η μέθοδος αυτή χρησιμοποιείται πρακτικά στους υπολογιστές, γιατί υλοποιείται πολύ πιο απλά απ' ότι ο γνωστός μας χειρωνακτικός τρόπος πολλαπλασιασμού. Πιο συγκεκριμένα, απαιτεί πολλαπλασιασμό επί δύο, διαίρεση διά δύο και πρόσθεση. Σε αντίθεση η γνωστή μας διαδικασία πολλαπλασιασμού απαιτεί πολλαπλασιασμό με οποιοδήποτε ακέραιο και πρόσθεση. Σε επίπεδο, λοιπόν, κυκλωμάτων υπολογιστή ο πολλαπλασιασμός επί δύο και η διαίρεση διά δύο μπορούν να υλοποιηθούν ταχύτατα με μία απλή εντολή ολίσθησης (shift), σε αντίθεση με τον πολλαπλασιασμό με οποιοδήποτε ακέραιο που θεωρείται πιο χρονοβόρα διαδικασία. Το τελευταίο γεγονός είναι ο λόγος που ο πολλαπλασιασμός αλά ρωσικά είναι προτιμότερος απ' ότι ο χειρωνακτικός τρόπος πολλαπλασιασμού δύο ακεραίων.

Στη συνέχεια παρουσιάζεται ο αλγόριθμος πολλαπλασιασμού ακεραίων αλά ρωσικά με φυσική γλώσσα κατά βήματα.

Αλγόριθμος: Πολλαπλασιασμός δύο θετικών ακεραίων (αλά ρωσικά)	
Είσοδος:	Δύο ακέραιοι M1 και M2, όπου M1, M2 ≥ 1
Έξοδος:	Το γινόμενο P=M1*M2
Βήμα 1	Θέσε P=0
Βήμα 2	Αν M2>0, τότε πήγαινε στο Βήμα 3, αλλιώς πήγαινε στο Βήμα 7
Βήμα 3	Αν ο M2 είναι περιττός, τότε θέσε P=P+M1
Βήμα 4	Θέσε M1=M1*2
Βήμα 5	Θέσε M2=M2/2 (θεώρησε μόνο το ακέραιο μέρος)
Βήμα 6	Πήγαινε στο Βήμα 2
Βήμα 7	Τύπωσε τον P.

Ακολουθεί ο αλγόριθμος σε ψευδοκώδικα για το ίδιο πρόβλημα του πολλαπλασιασμού αλά ρωσικά.

```
Αλγόριθμος Πολλαπλασιασμός_αλά_ρωσικά
Δεδομένα // M1,M2 ακέραιοι //
P ← 0
Οσο M2 > 0 επανάλαβε
    Αν M2 mod 2 = 1 τότε P ← P+M1
    M1 ← M1*2
    M2 ← M2 div 2
Τέλος_επανάληψης
Αποτελέσματα // P, το γινόμενο των ακεραίων M1,M2 //
Τέλος Πολλαπλασιασμός_αλά_ρωσικά
```

Ανακεφαλαίωση

Στο κεφάλαιο αυτό έγινε η πρώτη γνωριμία με την έννοια του αλγορίθμου. Δόθηκαν οι απαραίτητοι ορισμοί που συνοδεύτηκαν με αρκετά παραδείγματα. Αλγόριθμος είναι η διαδικασία της λύσης ενός προβλήματος. Η παράσταση των αλγορίθμων μπορεί να γίνει με αρκετούς τρόπους, ωστόσο η έμφαση δόθηκε στην παράσταση με χρήση ψευδογλώσσας. Στο κεφάλαιο αυτό αναπτύχθηκαν οι κύριες αλγοριθμικές δομές, δηλαδή η ακολουθία, η επιλογή και η επανάληψη ή ανακύκλωση, που θα χρησιμοποιηθούν στους αλγορίθμους των επόμενων κεφαλαίων.

Λέξεις κλειδιά

Αλγόριθμος, ακολουθία, επιλογή, επανάληψη, διάγραμμα ροής, ψευδογλώσσα, εμφωλευμένος, βρόχος.

Ερωτήσεις - Θέματα για συζήτηση

1. Να δοθεί ο ορισμός του όρου αλγόριθμος.
2. Ποιά είναι τα κριτήρια που πρέπει να ικανοποιεί κάθε αλγόριθμος;
3. Υπό ποία πρίσματα η Πληροφορική επιστήμη μελετά τους αλγορίθμους;
4. Ποιά η διαφορά της θεωρητικής από την αναλυτική προσέγγιση στην επίλυση ενός προβλήματος με χρήση αλγορίθμου;
5. Περιγράψτε τους τρόπους περιγραφής και αναπαράστασης των αλγορίθμων.
6. Ποιές είναι οι βασικοί τύποι συνιστωσών/εντολών ενός αλγορίθμου ;
7. Να περιγραφεί η δομή της ακολουθίας και να δοθεί σε διάγραμμα ροής ένα παράδειγμα αυτής της αλγοριθμικής προσέγγισης.
8. Να περιγραφεί η δομή της επιλογής και να δοθεί με ακολουθία βημάτων ένα παράδειγμα αυτής της αλγοριθμικής προσέγγισης.
9. Να περιγραφεί η δομή των επαναληπτικών διαδικασιών και να δοθεί με ακολουθία βημάτων και με διάγραμμα ροής ένα παράδειγμα αυτής της αλγοριθμικής προσέγγισης.
10. Να περιγραφεί η δομή των διαδικασιών πολλαπλών επιλογών και να δοθεί με ακολουθία βημάτων και με διάγραμμα ροής ένα παράδειγμα αυτής της αλγοριθμικής προσέγγισης.
11. Να περιγραφεί η δομή των εμφωλευμένων διαδικασιών και να δοθεί με

ακολουθία βημάτων και με διάγραμμα ροής ένα παράδειγμα αυτής της αλγοριθμικής προσέγγισης.

12. Να περιγραφεί με ακολουθία βημάτων το πρόβλημα του 'πολλαπλασιασμού αλά ρωσικά'.
13. Ποιά η πρακτική σημασία του αλγορίθμου του 'πολλαπλασιασμού αλά ρωσικά' ; Πότε γίνεται χρήση αυτού του τρόπου πολλαπλασιασμού δύο ακεραίων ;

Βιβλιογραφία

1. Ν.Ιωαννίδης, Κ.Μαρινάκης, Σπ.Μπακογιάννης, *Δομημένη Σχεδίαση Προγράμματος*, Εκδόσεις Ελιξ, Αθήνα 1991.
2. Χρήστος Κοίλιας, *Δομές Δεδομένων και Οργανώσεις Αρχείων*, Εκδόσεις Νέων Τεχνολογιών, 1993, Αθήνα.
3. Ιωάννης Μανωλόπουλος, *Δομές Δεδομένων – μία Προσέγγιση με Pascal*, Εκδόσεις Art of Text, Θεσσαλονίκη, 1998.
4. Σκανδάλης κ.α. *Στοιχεία Θεωρίας Αλγορίθμων*, Πανεπιστημιακές Εκδόσεις Κρήτης, Κρήτη, 1990.
5. D. Brunskill and J. Turner, *Understanding Algorithms and Data Structures*, McGraw-Hill, 1996.
6. D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, Vol.1, 3rd edition, Addison Wesley, 1997.
7. M.A. Weiss, *Data Structures and Algorithm Analysis*, 2nd edition, Benjamin/Cummings, 1995

Διευθύνσεις Διαδικτύου

http://www.nist.gov/dads/

Κόμβος με ευρετήριο όρων για αλγορίθμους, Δομές Δεδομένων και Προβλήματα (Algorithms, Data Structures, and Problems Terms and Definitions for the CRC Dictionary of Computer Science, Engineering and Technology)

ΚΕΦΑΛΑΙΟ 3ο

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΑΛΓΟΡΙΘΜΟΙ

ΠΕΡΙΕΧΟΜΕΝΑ

- Δεδομένα
- Αλγόριθμοι + Δομές δεδομένων = Προγράμματα
- Πίνακες
- Στοίβα και ουρά
- Αναζήτηση και Ταξινόμηση
- Αναδρομή
- Άλλες δομές δεδομένων

Εισαγωγή

Εκτός από τους αλγορίθμους, σημαντική έννοια για την Πληροφορική είναι και η έννοια των "δεδομένων". Τα δεδομένα αποθηκεύονται στον υπολογιστή με τη βοήθεια των λεγόμενων "δομών δεδομένων". Θεωρώντας τους αλγόριθμους και τις δομές δεδομένων μία αδιάσπαστη ενότητα μπορεί να λεχθεί, ότι η ενότητα αυτή τελικά αποτελεί τη βάση ενός προγράμματος, που επιλύει ένα πρόβλημα. Στο κεφάλαιο αυτό γίνεται μία εισαγωγή στις σπουδαιότερες δομές δεδομένων και τις αντίστοιχες πράξεις που μπορούμε να κάνουμε με αυτές, όπως είναι η αναζήτηση, η εισαγωγή και η εξαγωγή στοιχείων, καθώς και η ταξινόμηση.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να αιτιολογούν τη σπουδαιότητα των δεδομένων για την επίλυση ενός προβλήματος,
- να επισημαίνουν την αδιάσπαστη ενότητα αλγόριθμων και δομών δεδομένων,
- να εκτελούν γενικές ασκήσεις και ασκήσεις αναζήτησης και ταξινόμησης με χρήση της δομής του πίνακα,
- να ορίζουν τις δομές της στοίβας και της ουράς με τις αντίστοιχες λειτουργίες,
- να ορίζουν την έννοια της αναδρομής και να εκτελούν απλές σχετικές ασκήσεις,
- να γνωρίζουν τις δομές της λίστας και του δένδρου.

Προερωτήσεις

- ✓ Έχεις ακούσει για τον όρο FIFO;
- ✓ Γνωρίζεις ότι μπορεί να εξομοιωθεί στον υπολογιστή μια ουρά ανθρώπων, τρένων ή προγραμμάτων;
- ✓ Υπάρχει δυνατότητα ταχύτερης αναζήτησης μιας πληροφορίας ανάμεσα σε πολλές και με ποιον τρόπο;
- ✓ Ξέρεις, αν υπάρχουν πολλές μέθοδοι για να ταξινομηθούν κάποια αντικείμενα;

3.1 Δεδομένα

Τα *δεδομένα* (data) είναι η αφαιρετική αναπαράσταση της πραγματικότητας και συνεπώς μία απλοποιημένη όψη της. Για παράδειγμα, έστω ένα αρχείο μαθητών ενός σχολείου. Τα χρήσιμα δεδομένα που αποθηκεύονται είναι το ονοματεπώνυμο, η ηλικία, το φύλο, η τάξη, το τμήμα κλπ., όχι όμως το βάρος, το ύψος κλπ. Τα δεδομένα, λοιπόν, είναι ακατέργαστα γεγονότα, και κάθε φορά η επιλογή τους εξαρτάται από τον τύπο του προβλήματος. Η συλλογή των ακατέργαστων δεδομένων και ο συσχετισμός τους δίνει ως αποτέλεσμα την *πληροφορία* (information). Δεν είναι εύκολο να δοθεί επακριβής ορισμός της έννοιας της πληροφορίας, αλλά μπορεί να θεωρηθεί ότι ο αλγόριθμος είναι το μέσο για την παραγωγή πληροφορίας από τα δεδομένα. Με βάση τις δεδομένες πληροφορίες λαμβάνονται διάφορες αποφάσεις και γίνονται ενέργειες. Στη συνέχεια αυτές οι ενέργειες παράγουν νέα δεδομένα, νέες πληροφορίες, νέες αποφάσεις, νέες ενέργειες κοκ. Η μέτρηση, η κωδικοποίηση, η μετάδοση της πληροφορίας αποτελεί αντικείμενο μελέτης ενός ιδιαίτερου κλάδου, της *Θεωρίας Πληροφοριών* (Information Theory), που είναι ένα ιδιαίτερα σημαντικό πεδίο της Πληροφορικής.

Όπως η Πληροφορική ορίζεται ως επιστήμη σε συνάρτηση με την έννοια του αλγορίθμου, κατά τον ίδιο τρόπο η Πληροφορική ορίζεται και σε σχέση με την έννοια των δεδομένων. Έτσι, Πληροφορική θεωρείται η επιστήμη που μελετά τα δεδομένα από τις ακόλουθες σκοπιές:

- **Υλικού.** Το υλικό (hardware), δηλαδή η μηχανή, επιτρέπει στα δεδομένα ενός προγράμματος να αποθηκεύονται στην κύρια μνήμη και στις περιφερειακές συσκευές του υπολογιστή με διάφορες αναπαραστάσεις (representations). Τέτοιες μορφές είναι η δυαδική, ο κώδικας ASCII (βλ. παράρτημα), ο κώδικας EBCDIC, το συμπλήρωμα του 1 ή του 2 κ.λπ.
- **Γλωσσών προγραμματισμού**. Οι γλώσσες προγραμματισμού υψηλού επιπέδου (high level programming languages) επιτρέπουν τη χρήση διάφορων τύπων (types) μεταβλητών (variables) για να περιγράψουν ένα δεδομένο. Ο μεταφραστής κάθε γλώσσας φροντίζει για την αποδοτικότερη μορφή αποθήκευσης, από πλευράς υλικού, κάθε μεταβλητής στον υπολογιστή.
- **Δομών Δεδομένων**. Δομή δεδομένων (data structure) είναι ένα σύνολο δεδομένων μαζί με ένα σύνολο επιτρεπτών λειτουργιών επί αυτών.Για παράδειγμα, μία τέτοια δομή είναι η εγγραφή (record), που μπορεί να περιγράφει ένα είδος, ένα πρόσωπο κλπ. Η εγγραφή αποτελείται από τα πεδία (fields) που αποθηκεύουν χαρακτηριστικά (attributes) διαφορετικού τύπου, όπως για παράδειγμα ο κωδικός, η περιγραφή κλπ. 'Αλ-

Μια θέση μνήμης (byte) έχει ως περιεχόμενο 11110001. Η τιμή μπορεί να παριστάνει:

- *Το χαρακτήρα _ στον κώδικα ASCII 437*
- *Το χαρακτήρα ρ στον κώδικα ΕΛΟΤ 928*
- *Το χαρακτήρα 1 στον κώδικα EBCDIC*
- *Την τιμή 241 στο δυαδικό σύστημα (ως μη προσημασμένο ακέραιο)*
- *Την τιμή -14 στο δυαδικό σύστημα (ως προσημασμένο ακέραιο στο συμπλήρωμα ως προς 1)*
- *Την τιμή -15 στο δυαδικό σύστημα (ως προσημασμένο ακέραιο στο συμπλήρωμα ως προς 2)*

Ακόμη μπορεί να είναι τμήμα ενός ακέραιου σε 2 ή 4 bytes, καθώς και ενός αριθμού κινητής υποδιαστολής.

Οσον αφορά στη φυσική σημασία της, αν μεν είναι χαρακτήρας, τότε αποτελεί μέρος μιας αλφαριθμητικής σταθεράς, ενώ αν πρόκειται για αριθμητική τιμή, τότε μπορεί να είναι δεδομένο, διεύθυνση μνήμης ή κώδικας εντολής προγράμματος.

λη μορφή δομής δεδομένων είναι το αρχείο που αποτελείται από ένα σύνολο εγγραφών. Μία επιτρεπτή λειτουργία σε ένα αρχείο είναι η σειριακή προσπέλαση όλων των εγγραφών του.

- **Ανάλυσης Δεδομένων**. Τρόποι καταγραφής και αλληλοσυσχέτισης των δεδομένων μελετώνται έτσι ώστε να αναπαρασταθεί η γνώση για πραγματικά γεγονότα. Οι τεχνολογίες των Βάσεων Δεδομένων (Databases), της Μοντελοποίησης Δεδομένων (Data Modelling) και της Αναπαράστασης Γνώσης (Knowledge Representation) ανήκουν σε αυτή τη σκοπιά μελέτης των δεδομένων.

3.2 Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα

Τα δεδομένα ενός προβλήματος αποθηκεύονται στον υπολογιστή, είτε στην κύρια μνήμη του είτε στη δευτερεύουσα μνήμη του. Η αποθήκευση αυτή δεν γίνεται κατά ένα τυχαίο τρόπο αλλά συστηματικά, δηλαδή χρησιμοποιώντας μία δομή. Η έννοια της *δομής δεδομένων* (data structure) είναι σημαντική για την Πληροφορική και ορίζεται με τον ακόλουθο τυπικό ορισμό.

Ορισμός: Δομή Δεδομένων είναι ένα σύνολο αποθηκευμένων δεδομένων που υφίστανται επεξεργασία από ένα σύνολο λειτουργιών.

Κάθε μορφή δομής δεδομένων αποτελείται από ένα σύνολο κόμβων (nodes). Οι βασικές λειτουργίες (ή αλλιώς πράξεις) επί των δομών δεδομένων είναι οι ακόλουθες:

- ✓ ***Προσπέλαση*** (access), πρόσβαση σε ένα κόμβο με σκοπό να εξετασθεί ή να τροποποιηθεί το περιεχόμενό του.
- ✓ ***Εισαγωγή*** (insertion), δηλαδή η προσθήκη νέων κόμβων σε μία υπάρχουσα δομή.
- ✓ ***Διαγραφή*** (deletion), που αποτελεί το αντίστροφο της εισαγωγής, δηλαδή ένας κόμβος αφαιρείται από μία δομή.
- ✓ ***Αναζήτηση*** (searching), κατά την οποία προσπελαύνονται οι κόμβοι μιας δομής, προκειμένου να εντοπιστούν ένας ή περισσότεροι που έχουν μια δεδομένη ιδιότητα.
- ✓ ***Ταξινόμηση*** (sorting), όπου οι κόμβοι μιας δομής διατάσσονται κατά αύξουσα ή φθίνουσα σειρά.

- ✓ ***Αντιγραφή*** (copying), κατά την οποία όλοι οι κόμβοι ή μερικοί από τους κόμβους μίας δομής αντιγράφονται σε μία άλλη δομή.
- ✓ ***Συγχώνευση*** (merging), κατά την οποία δύο ή περισσότερες δομές συνενώνονται σε μία ενιαία δομή.
- ✓ ***Διαχωρισμός*** (separation), που αποτελεί την αντίστροφη πράξη της συγχώνευσης.

Στην πράξη σπάνια χρησιμοποιούνται και οι οκτώ λειτουργίες για κάποια δομή. Συνηθέστατα παρατηρείται το φαινόμενο μία δομή δεδομένων να είναι αποδοτικότερη από μία άλλη δομή με κριτήριο κάποια λειτουργία, για παράδειγμα την αναζήτηση, αλλά λιγότερο αποδοτική για κάποια άλλη λειτουργία, για παράδειγμα την εισαγωγή. Αυτές οι παρατηρήσεις εξηγούν αφ' ενός την ύπαρξη διαφορετικών δομών, και αφ' ετέρου τη σπουδαιότητα της επιλογής της κατάλληλης δομής κάθε φορά.

Στη συνέχεια του βιβλίου αυτού θα γίνει πληρέστερη παρουσίαση εναλλακτικών δομών δεδομένων. Ωστόσο, στο σημείο αυτό τονίζεται ότι υπάρχει μεγάλη εξάρτηση μεταξύ της δομής δεδομένων και του αλγόριθμου που επεξεργάζεται τη δομή. Μάλιστα, το πρόγραμμα πρέπει να θεωρεί τη δομή δεδομένων και τον αλγόριθμο ως μία αδιάσπαστη ενότητα. Η παρατήρηση αυτή δικαιολογεί την εξίσωση που διατυπώθηκε το 1976 από τον Wirth (που σχεδίασε και υλοποίησε τη γλώσσα Pascal)

Αλγόριθμοι + Δομές Δεδομένων = Προγράμματα

Ωστόσο για την πληρέστερη κατανόηση της σχέσης αυτής στη συνέχεια θα εξετασθεί ένα τέτοιο πρόβλημα.

Παράδειγμα

Έστω ότι πρέπει να γραφεί ένας αλγόριθμος που να δέχεται ως είσοδο το όνομα ενός συνδρομητή του ΟΤΕ και να δίνει ως έξοδο το τηλέφωνό του.

Πρώτη Λύση.

Δομή Δεδομένων: Δημιουργείται μία ακολουθία (O_1,T_1), (O_2,T_2), ..., (O_n,T_n), όπου οι μεταβλητές O_i και T_i αναφέρονται στο όνομα και στο τηλέφωνο του i-οστού συνδρομητή, για $i=1,2,...,n$.

Αλγόριθμος: Η ακολουθία ανιχνεύεται μέχρι να βρεθεί το ζητούμενο όνομα του συνδρομητή O_k και εκτυπώνεται το τηλέφωνο T_k. Ο αλγόριθμος αυτός είναι αποδοτικός για συνδρομητές ενός χωριού ή μίας κωμόπολης, αλλά για συνδρομητές μίας μεγάλης πόλης είναι χρονοβόρος.

Δεύτερη Λύση.

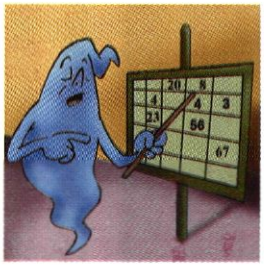
Δομή Δεδομένων: Χρησιμοποιείται και πάλι η ακολουθία της πρώτης λύσης, αλλά αυτή τη φορά οι συνδρομητές είναι ταξινομημένοι λεξικογραφικά. Επιπλέον δημιουργείται μία δεύτερη ακολουθία με τα στοιχεία (A,n_1), (B,n_2), ..., (Ω,n_{24}). Κάθε στοιχείο της δεύτερης αυτής ακολουθία δίνει για κάθε γράμμα του αλφαβήτου τη θέση n_i (για $i=1, 2, ..., 24$) στην πρώτη ακολουθία με το πρώτο όνομα συνδρομητή που αρχίζει από το γράμμα αυτό.

Αλγόριθμος: Αφήνεται για άσκηση στο μαθητή.

Οι δομές δεδομένων διακρίνοναι σε δύο μεγάλες κατηγορίες: τις *στατικές* (static) και τις *δυναμικές* (dynamic). Οι δυναμικές δομές δεν αποθηκεύονται σε συνεχόμενες θέσεις μνήμης αλλά στηρίζονται στην τεχνική της λεγόμενης *δυναμικής παραχώρησης μνήμης* (dynamic memory allocation). Με άλλα λόγια, οι δομές αυτές δεν έχουν σταθερό μέγεθος, αλλά ο αριθμός των κόμβων τους μεγαλώνει και μικραίνει καθώς στη δομή εισάγονται νέα δεδομένα ή διαγράφονται κάποια δεδομένα αντίστοιχα. Όλες οι σύγχρονες γλώσσες προγραμματισμού προσφέρουν τη δυνατότητα δυναμικής παραχώρησης μνήμης. Ωστόσο, εμείς στη συνέχεια θα εξετάσουμε μόνο τις στατικές δομές που είναι ευκολότερες στην κατανόηση και την υλοποίησή τους

3.3 Πίνακες

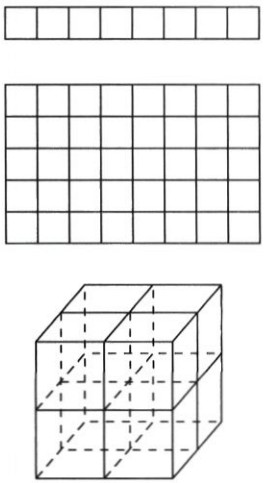
Με τον όρο *στατική δομή δεδομένων* εννοείται ότι το ακριβές μέγεθος της απαιτούμενης κύριας μνήμης καθορίζεται κατά τη στιγμή του προγραμματισμού τους, και κατά συνέπεια κατά τη στιγμή της μετάφρασής τους και όχι κατά τη στιγμή της εκτέλεσης τους προγράμματος. Μία άλλη σημαντική διαφορά σε σχέση με τις δυναμικές δομές είναι ότι τα στοιχεία των στατικών δομών αποθηκεύονται σε συνεχόμενες θέσεις μνήμης.



Στην πράξη, οι στατικές δομές υλοποιούνται με πίνακες που μας είναι γνωστοί από άλλα μαθήματα και υποστηρίζονται από κάθε γλώσσα προγραμματισμού. Μπορούμε να ορίσουμε τον πίνακα ως μια δομή που περιέχει στοιχεία του ίδιου τύπου (δηλαδή ακέραιους, πραγματικούς κ.λπ). Η δήλωση των στοιχείων ενός πίνακα και η μέθοδος αναφοράς τους εξαρτάται από τη συγκεκριμένη γλώσσα υψηλού επιπέδου που χρησιμοποιείται. Όμως, γενικά η αναφορά στα στοιχεία ενός πίνακα γίνεται με τη χρήση του συμβολικού ονόματος του πίνακα ακολουθούμενου από την τιμή ενός ή περισσότερων *δεικτών* (indexes) σε παρένθεση ή αγκύλη.

Ένας πίνακας μπορεί να είναι μονοδιάστατος, αλλά στη γενικότερη πε-

ρίπτωση μπορεί να είναι δισδιάστατος, τρισδιάστος και γενικά ν-διάστατος πίνακας. Όσον αφορά στους δισδιάστους πίνακες σημειώνεται ότι αν το μέγεθος των δύο διαστάσεων είναι ίσο, τότε ο πίνακας λέγεται *τετραγωνικός* (square) και γενικά συμβολίζεται ως πίνακας n x n. Μάλιστα μορούμε να θεωρήσουμε το δισδιάστο πίνακα ότι είναι ένας μονοδιάστατος πίνακας, όπου κάθε θέση του περιέχει ένα νέο μονοδιάστατο πίνακα. Στη συνέχεια δίνουμε δύο απλά παραδείγματα χρήσης πινάκων, τα οποία στηρίζονται σε αλγορίθμους του προηγουμένου κεφαλαίου.



***Σχ. 3.1** Παραδείγματα πινάκων (μονοδιάστατος, δισδιάστατος, τριδιάστατος)*

Παράδειγμα 1. Εύρεση του μικρότερου στοιχείου ενός μονοδιάστατου πίνακα

Δίνεται ένας μονοδιάστατος πίνακα table 100 στοιχείων. Να σχεδιασθεί αλγόριθμος που να βρίσκει το μικρότερο στοιχείο του

```
Αλγόριθμος Ελάχ_Πίνακα
Δεδομένα // table //
Min ← table[1]
Για i από 2 μέχρι 100
    Αν table[i] < Min τότε Min ← table[i]
Τέλος_επανάληψης
Αποτελέσματα //Min//
Τέλος Ελάχ_Πίνακα
```

Στον αλγόριθμο αυτό αρχικά το πρώτο στοιχείο του πίνακα εκχωρείται στη μεταβλητή Min. Στη συνέχεια κάθε επόμενο στοιχείο του πίνακα εξετάζεται, αν είναι μικρότερο της Min και αν ναι, τότε αντικαθιστά το προηγούμενο. Ετσι στο τέλος θα υπάρχει στη μεταβλητή Min το μικρότερο στοιχείο όλου του πίνακα table.

Παράδειγμα 2. Εύρεση αθροίσματος στοιχείων δισδιάστατου πίνακα

Δίδεται ο δισδιάστατος πίνακας table με m γραμμές n στήλες. Να βρεθεί το άθροισμα κατά γραμμή, κατά στήλη και συνολικά.

Στη συνέχεια ακολουθεί ο αλγόριθμος που επιλύει το πρόβλημα. Για καλύτερη κατανόηση σημειώνεται, ότι οι δύο πρώτοι βρόχοι μηδενίζουν τις αντίστοιχες μεταβλητές που θα υποδεχθούν τα αθροίσματα. Αυτό είναι μία τακτική που πρέπει να εφαρμόζεται οποτεδήποτε στα προβλήματά μας έχουμε να υπολογίσουμε αθροίσματα.

```
Αλγόριθμος Αθρ_Πίνακα
Δεδομένα // m, n, table //
sum ← 0
Για i από 1 μέχρι m
    row[i] ← 0
Τέλος_επανάληψης
Για j από 1 μέχρι n
    col[j] ← 0
Τέλος_επανάληψης
Για i από 1 μέχρι m
    Για j από 1 μέχρι n
        sum ← sum + table[i,j]
        row[i] ← row[i] + table[i,j]
        col[j] ← col[j] + table[i,j]
    Τέλος_επανάληψης
Τέλος_επανάληψης
Αποτελέσματα // row, col, sum //
Τέλος Αθρ_Πίνακα
```

Ο διπλός εμφωλευμένος βρόχος που ακολουθεί τους δύο πρώτους απλούς βρόχους, είναι η καρδιά του αλγορίθμου, όπου γίνονται οι υπολογισμοί που ζητά η εκφώνηση του παραδείγματος. Γενικά σε εμφωλευμένους βρόχους, μία τιμή μεταβλητής του εξωτερικού βρόχου παραμένει σταθερή, όσο μεταβάλλεται η τιμή της μεταβλητής του εσωτερικού βρόχου. Πιο συγκεκριμένα, στον αλγόριθμό μας αρχικά το i λαμβάνει την τιμή 1 και το j διαδοχικά τις τιμές 1,2,...,n. Κατόπιν, το i λαμβάνει την τιμή 2, ενώ το j και πάλι λαμβάνει διαδοχικά τις τιμές 1,2,...,n. Η διαδικασία αυτή επαναμβάνεται μέχρι το i να λάβει την τιμή m.

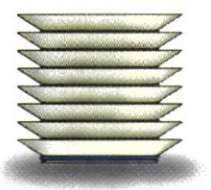
Ο επόμενος πίνακας είναι ένας δισδιάστατος πίνακας 5x5. Αν ο προηγούμενος αλγόριθμος εφαρμοσθεί στον πίνακα αυτό, τότε οι τιμές των στοιχείων του πίνακα row παρουσιάζονται στην τελευταία κατακόρυφη στήλη, ενώ οι τιμές των στοιχείων του πίνακα col παρουσιάζονται στην τελευταία γραμμή του πίνακα. Τέλος το συνολικό άθροισμα sum παρουσιάζεται στην κάτω-δεξιά γωνία.

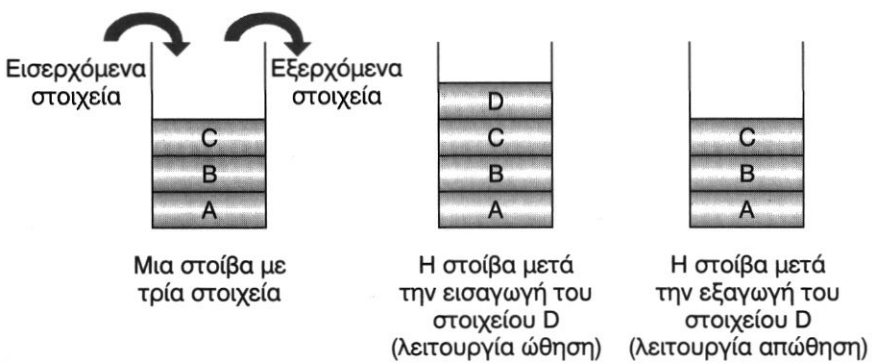
	Πίνακας table					Πίνακας row	
	4	16	5	21	7	53	
	28	9	38	13	51	139	
	17	67	22	40	30	176	
	20	40	10	3	13	86	
	21	34	48	29	26	158	
Πίνακας col	90	166	123	106	127	612	Sum

Οι πίνακες χρησιμεύουν για την αποθήκευση και διαχείριση δύο σπουδαίων δομών, της στοίβας (stack) και της ουράς (queue), που θα εξεταστούν λεπτομερέστερα στη συνέχεια, επειδή χρησιμοποιούνται σε πληθώρα πρακτικών εφαρμογών.

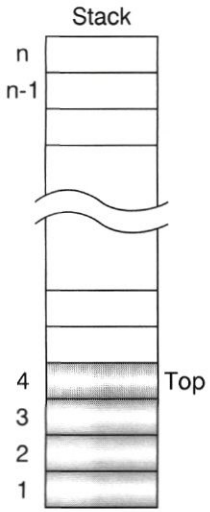
3.4 Στοίβα

Μία στοίβα δεδομένων μοιάζει με μία στοίβα από πιάτα. Για παράδειγμα, κάθε πιάτο που πλένεται τοποθετείται στην κορυφή (top) της στοίβας των πιάτων, ενώ για σκούπισμα λαμβάνεται και πάλι το πιάτο της κορυφής. Αντίστοιχα, τα δεδομένα που βρίσκονται στην κορυφή της στοίβας λαμβάνονται πρώτα, ενώ αυτά που βρίσκονται στο βάθος της στοίβας λαμβάνονται τελευταία. Αυτή η μέθοδος επεξεργασίας ονομάζεται *Τελευταίο μέσα, πρώτο έξω* ή απλούστερα με την αγγλική συντομογραφία LIFO





Σχ. 3.2. Λειτουργίες στοίβας.



Σχ. 3.3 Υλοποίηση στοίβας με χρήση πίνακα

(Last-In-First-Out). Στα αριστερά του επόμενου σχήματος δίνεται μία στοίβα με τρία στοιχεία, ενώ στο κέντρο παρουσιάζεται η ίδια στοίβα με ένα πρόσθετο στοιχείο στην κορυφή της.

Δύο είναι οι κύριες λειτουργίες σε μία στοίβα:

- η *ώθηση* (push) στοιχείου στην κορυφή της στοίβας, και
- η *απώθηση* (pop) στοιχείου από τη στοίβα.

Η διαδικασία της ώθησης πρέπει οπωσδήποτε να ελέγχει, αν η στοίβα είναι γεμάτη, οπότε λέγεται ότι συμβαίνει *υπερχείλιση* (overflow) της στοίβας. Αντίστοιχα, η διαδικασία απώθησης ελέγχει, αν υπάρχει ένα τουλάχιστον στοιχείο στη στοίβα, δηλαδή ελέγχει αν γίνεται *υποχείλιση* (underflow) της στοίβας.

Μια στοίβα μπορεί να υλοποιηθεί πολύ εύκολα με τη βοήθεια ενός μονοδιάστατου πίνακα, όπως φαίνεται στο σχήμα 3.3. Μια βοηθητική μεταβλητή (με όνομα συνήθως top) χρησιμοποιείται για να δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας. Για την εισαγωγή ενός νέου στοιχείου στη στοίβα (ώθηση) αρκεί να αυξηθεί η μεταβλητή top κατά ένα και στη θέση αυτή να εισέλθει το στοιχείο. Αντίθετα για την εξαγωγή ενός στοιχείου από τη στοίβα (απώθηση) εξέρχεται πρώτα το στοιχείο που δείχνει η μεταβλητή top και στη συνέχεια η top μειώνεται κατά ένα για να δείχνει τη νέα κορυφή.

3.5 Ουρά

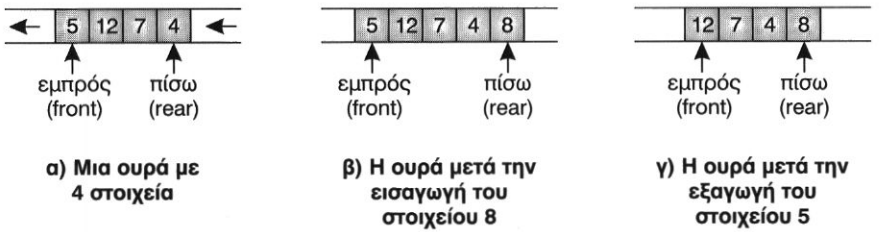
Οι ουρές είναι καθημερινό φαινόμενο. Για παράδειγμα, ουρές δημιουργούνται όταν άνθρωποι, αυτοκίνητα, εργασίες, προγράμματα κ.λπ. περιμένουν για να εξυπηρετηθούν. Το θέμα είναι τόσο σημαντικό και με τέτοιες πρακτικές επιπτώσεις, ώστε ένας ιδιαίτερος κλάδος των Μαθηματικών, η Επιχειρησιακή Έρευνα (Operations Research), και ιδιαίτερα η Θεωρία Ουρών (Queueing Theory), μελετά τη συμπεριφορά και την επίδοση των ουρών. Σε μία ουρά αναμονής με ανθρώπους, συμβαίνει να εξυπηρετείται εκείνος που στάθηκε στην ουρά πρώτος από όλους τους άλλους (αν και υπάρχουν εξαιρέσεις που όμως δεν θα εξετασθούν στο βιβλίο αυτό). Η μέθοδος αυτή επεξεργασίας ονομάζεται *Πρώτο μέσα, πρώτο έξω* ή απλούστερα ακολουθώντας την αγγλική συντομογραφία FIFO (First-In-First-Out).

Δύο είναι οι κύριες λειτουργίες που εκτελούνται σε μία ουρά:

- η *εισαγωγή* (enqueue) στοιχείου στο πίσω άκρο της ουράς, και
- η εξαγωγή (dequeue) στοιχείου από το εμπρός άκρο της ουράς.

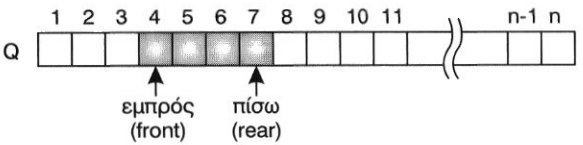
Άρα, σε αντίθεση με τη δομή της στοίβας, στην περίπτωση της ουράς απαιτούνται δύο δείκτες: ο *εμπρός* (front) και ο *πίσω* (rear) δείκτης, που μας δίνουν τη θέση του στοιχείου που σε πρώτη ευκαιρία θα εξαχθεί και τη θέση του στοιχείου που μόλις εισήλθε.

Στο σχήμα 3.4 φαίνεται μια ουρά με τέσσερα στοιχεία (α), στην οποία εισάγεται ένα νέο στοιχείο (β) και ακολούθως εξάγεται ένα στοιχείο.



Σχ. 3.4. *Εισαγωγή και εξαγωγή από ουρά.*

Μια ουρά μπορεί να υλοποιηθεί με τη βοήθεια ενός μονοδιάστατου πίνακα, όπως φαίνεται στο σχήμα 3.5. Για την εισαγωγή ενός νέου στοιχείου στην ουρά αυξάνεται ο δείκτης rear κατά ένα και στη θέση αυτή αποθηκεύεται το στοιχείο. Αντίστοιχα για τη λειτουργία της εξαγωγής, εξέρχεται το στοιχείο που δείχνει ο δείκτης front, ο οποίος στη συνέχεια αυξάνεται κατά ένα, για να δείχνει το επόμενο στοιχείο που πρόκειται να εξαχθεί. Σε κάθε περίπτωση όμως, πρέπει να ελέγχεται πριν από οποιαδήποτε ενέργεια, αν υπάρχει ελεύθερος χώρος στον πίνακα για την εισαγωγή και αν υπάρχει ένα τουλάχιστον στοιχείο για την εξαγωγή.



Σχ. 3.5 *Υλοποίηση ουράς με χρήση πίνακα*

FIFO και LIFO

Όπως είδαμε η δομή της στοίβας λειτουργεί με τη μέθοδο FIFO. Οι δύο αυτές μέθοδοι έχουν αρκετές χρήσεις σε πραγματικά προβλήματα. Ας θεωρήσουμε για παράδειγμα την περίπτωση ενός αποθηκευτικού χώρου μιας επιχείρησης. Σε κάθε αποθήκη γίνονται εισαγωγές ειδών που προέρχονται από αγορές από προμηθευτές, αν η επιχείρηση είναι εμπορική ή από την παραγωγή, αν πρόκειται για βιομηχανική επιχείρηση. Τα εμπορεύματα ή προϊόντα τοποθετούνται σε κάποιους χώρους, αποθήκες, ράφια κ.λπ. Οταν γίνονται πωλήσεις κάποιων ειδών, τα είδη αυτά βγαίνουν από την αποθήκη και αποστέλονται στους πελάτες. Έτσι εισαγωγές και εξαγωγές ειδών γίνονται συνεχώς στην αποθήκη ανάλογα με τη διαδικασία προμηθειών και τη ροή των πωλήσεων.

Σε μια δεδομένη στιγμή για κάποιο είδος μπορεί να υπάρχουν αποθηκευμένα κάποια τεμάχια που προέρχονται από μια παραλαβή και κάποια άλλα που υπήρχαν πιο πριν. Όταν πρέπει να εξαχθεί λοιπόν ένα τεμάχιο από αυτό το είδος, προκύπτει το πρόβλημα, από ποια παρτίδα πρέπει να είναι;

Η απάντηση στο ερώτημα αυτό έχει φυσική και λογιστική αξία. Αν το είδος αυτό δεν επηρεάζεται από το χρόνο, τότε ίσως δεν έχει μεγάλη σημασία η επιλογή. Αν όμως πρόκειται για είδος που μπορεί να αλλοιωθεί ή έχει ημερομηνία λήξης (π.χ. φάρμακα), τότε είναι φανερό ότι πρέπει να επιλεγεί το παλαιότερο. Στην περίπτωση αυτή λοιπόν πρέπει η εξαγωγή των ειδών να γίνεται με τη μέθοδο FIFO και συνήθως επαφίεται στον αποθηκάριο να κάνει τη σωστή επιλογή.

Εξ ίσου δύσκολο είναι το πρόβλημα αυτό από την οικονομική και λογιστική σκοπιά, που μάλιστα αφορά όλα τα είδη με ή χωρίς ημερομηνία λήξης. Ας υποθέσουμε ότι μια επιχείρηση έχει πραγματοποιήσει τις επόμενες αγορές και πωλήσεις για ένα είδος.

Αγορές

Ημ/νία	Ποσότητα	Τιμή μονάδας	Αξία
1/1/02	4	10	40
15/1/02	6	12	72
ΣΥΝΟΛΟ	10		112

Πωλήσεις

Ημ/νία	Ποσότητα	Τιμή μονάδας	Αξία
30/1/02	5	20	100

Από τα παραπάνω στοιχεία αγορών και πωλήσεων δημιουργείται η επόμενη καρτέλα είδους.

Καρτέλα είδους

Ημ/νία	Αιτιολογία	Ποσότητα			Αξία κόστους		
		Εισαγωγή	Εξαγωγή	Υπόλοιπο	Εισαγωγή	Εξαγωγή	Υπόλοιπο
1/1/02	Αγορά	4		4	40		40
15/1/02	Αγορά	6		10	72		112
30/1/02	Πώληση		5	5		X	Y

Το πρόβλημα που ανακύπτει στις εφαρμογές αυτές είναι ο καθορισμός των τιμών X και Y. Από τις τιμές αυτές εξάγεται στη συνέχεια το καθαρό κέρδος, με το οποίο η επιχείρηση θα φορολογηθεί.

α) Λειτουργία LIFO

Στις 30/1/02 τα 5 τεμάχια που πουλήθηκαν θεωρούνται ότι ανήκουν στα 6 τεμάχια της τελευταίας αγοράς, δηλαδή με τιμή μονάδας 12€. Άρα X = 5x12 = 60€ και Y = 112-52 = 60. Τώρα, το καθαρό κέρδος της πώλησης γίνεται 100-60 = 40.

β) Λειτουργία FIFO

Στις 30/1/02 από τα 5 τεμάχια που πουλήθηκαν, τα 4 είναι από την αγορά της 1/1/02 και το 1 από την αγορά της 15/1/02. Άρα το κόστος τους είναι X = 4x10+1x12 = 52 και Y = 112-52 = 60. Τώρα, το καθαρό κέρδος της πώλησης γίνεται 100-52 = 48.

γ) Λειτουργία με τη σταθμική μέση τιμή

Λόγω της αυξημένης πολυπλοκότητας των αντίστοιχων προγραμμάτων, αλλά και των απαιτούμενων διαδικασιών οι περισσότερες επιχειρήσεις εφαρμόζουν τη μέθοδο της **σταθμικής μέσης τιμής**. Η τελευταία για το προηγούμενο παράδειγμα είναι 112/10 = 11,2. Άρα X = 5x11,2 = 56 και Y = 112-56 = 56. Στην περίπτωση αυτή το καθαρό κέρδος γίνεται 100-56 = 44€.

3.6 Αναζήτηση

Το πρόβλημα της *αναζήτησης* (searching) ενός στοιχείου σε πίνακα είναι ιδιαίτερα ενδιαφέρον λόγω της χρησιμότητάς του σε πλήθος εφαρμογών. Υπάρχουν αρκετές μέθοδοι αναζήτησης σε πίνακα που εξαρτώνται κυρίως από το, αν ο πίνακας είναι ταξινομημένος ή όχι. Μια άλλη παράμετρος είναι, αν ο πίνακας περιέχει στοιχεία που είναι όλα διάφορα μεταξύ τους ή όχι. Τα στοιχεία του πίνακα μπορεί να είναι αριθμητικά ή αλφαριθμητικά.

Η πιο απλή μορφή αναζήτησης στοιχείου σε πίνακα είναι η *σειριακή* (sequential) ή *γραμμική* (linear) μέθοδος. Έτσι για τον επόμενο αλγόριθμο Sequential Search υποτίθεται ότι αναζητείται η τιμή key στο μη ταξινομημένο πίνακα table. Μετά την εκτέλεση του αλγορίθμου η μεταβλητή position επιστρέφει την τιμή 0, αν η αναζήτηση είναι ανεπιτυχής, ενώ αν η αναζήτηση είναι επιτυχής, τότε επιστρέφει τη θέση του στοιχείου στον πίνακα (δηλαδή, έναν αριθμό από 1 ως n).

```
Αλγόριθμος Sequential_Search
Δεδομένα // n, table, key //
done ← ψευδής
position ← 0
i ← 1
Όσο (done=ψευδής) και (i<=n) επανάλαβε
Αν table[i]=key τότε
    done ← αληθής
    position ← i
αλλιώς
    i ← i+1
Τέλος_αν
Τέλος_επανάληψης
Αποτελέσματα //done, position //
Τέλος Sequential_Search
```

Όπως αναφέρθηκε, τα στοιχεία που περιέχονται στον πίνακα table δεν είναι ταξινομημένα. Επίσης, ο προηγούμενος αλγόριθμος ισχύει για την περίπτωση όπου κάθε στοιχείο υπάρχει μία μόνο φορά στον πίνακα. Αν κάποιο στοιχείο εμφανίζεται στον πίνακα περισσότερο από μία φορές, τότε ο αλγόριθμος πρέπει να τροποποιηθεί κατά το εξής: η μεταβλητή done είναι περιττή και η αναζήτηση συνεχίζεται μέχρι το τέλος του πίνακα ελέγχοντας με τη συνθήκη $i \leq n$. Εξ άλλου, αν τα στοιχεία του πίνακα είναι ταξινομημένα, τότε ο αλγόριθμος πρέπει να σταματήσει, μόλις συναντήσει κάποιο στοιχείο που είναι μεγαλύτερο από το αναζητούμενο.

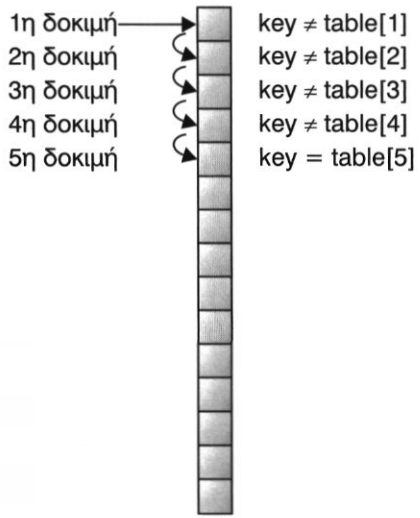
1	2	3	4	5	6	7	8	9
52	12	71	56	5	10	19	90	45

Για παράδειγμα, στο προηγούμενο σχήμα παρουσιάζεται ένας πίνακας που περιέχει εννέα αταξινόμητους ακεραίους. Έτσι, για την επιτυχή αναζήτηση της τιμής 56 απαιτούνται 4 προσπελάσεις. Αντίθετα, για την αναζήτηση της (ανύπαρκτης) τιμής 11 απαιτούνται 9 προσπελάσεις στον πίνακα, δηλαδή σάρωση ολόκληρου του πίνακα. Στο επόμενο σχήμα παρουσιάζεται ένας πίνακας που περιέχει τα ίδια στοιχεία αλλά σε ταξινομημένη μορφή. Στον πίνακα αυτό η ανεπιτυχής αναζήτηση για την τιμή 11 τερματίζει μετά την τρίτη προσπάθεια και την ανάγνωση του αριθμού 12.

1	2	3	4	5	6	7	8	9
5	10	12	19	45	52	56	71	90

Η σειριακή μέθοδος αναζήτησης είναι η πιο απλή, αλλά και η λιγότερη αποτελεσματική μέθοδος αναζήτησης. Έτσι, δικαιολογείται η χρήση της μόνο σε περιπτώσεις όπου:

- ✓ ο πίνακας είναι μη ταξινομημένος,
- ✓ ο πίνακας είναι μικρού μεγέθους (για παράδειγμα, $n \leq 20$),
- ✓ η αναζήτηση σε ένα συγκεκριμένο πίνακα γίνεται σπάνια,



Σχ. 3.6. Σειριακή αναζήτηση

Σε επόμενο κεφάλαιο θα εξετασθεί μία αποτελεσματικότερη μέθοδος αναζήτησης, η δυαδική αναζήτηση.

3.7 Ταξινόμηση

Η τακτοποίηση των κόμβων μίας δομής με μία ιδιαίτερη σειρά είναι μία πολύ σημαντική λειτουργία που ονομάζεται *ταξινόμηση* (sorting) ή *διάταξη* (ordering). Συνήθως η σειρά αυτή είναι η *αύξουσα τάξη* (ascending sequence) της τιμής των μεγεθών προς ταξινόμηση. Από το προηγούμενο παράδειγμα έγινε σαφές ότι σκοπός της ταξινόμησης είναι να διευκολυνθεί στη συνέχεια η αναζήτηση των στοιχείων του ταξινομημένου πίνακα. Η χρησιμότητα της ταξινόμησης αποδεικνύεται στην πράξη σε αναρίθμητες περιπτώσεις αναζήτησης αριθμητικών ή αλφαβητικών δεδομένων, όπως σε βιβλιοθηκονομικά συστήματα, λεξικά, τηλεφωνικούς καταλόγους, καταλόγους φόρου εισοδήματος και γενικά παντού όπου γίνεται αναζήτηση αποθηκευμένων αντικειμένων. Στη συνέχεια δίνεται ένας τυπικός ορισμός της ταξινόμησης.

Ορισμός. Δοθέντων των στοιχείων $a_1,a_2,\ldots,a_n$ η ταξινόμηση συνίσταται στη *μετάθεση* (permutation) της θέσης των στοιχείων, ώστε να τοποθετηθούν σε μία σειρά $a_{k1},a_{k2},\ldots,a_{kn}$ έτσι ώστε, δοθείσης μίας *συνάρτησης διάταξης* (ordering function), f, να ισχύει:

$$f(a_{k1}) \le f(a_{k2}) \le \ldots \le f(a_{kn})$$

Αξίζει να σημειωθεί ότι η προηγούμενη συνάρτηση διάταξης μπορεί να τροποποιηθεί, ώστε να καλύπτει και την περίπτωση που η ταξινόμηση γίνεται με *φθίνουσα τάξη* (descending sequence) μεγέθους.

Ταξινόμηση ευθείας ανταλλαγής

Η μέθοδος της *ταξινόμησης ευθείας ανταλλαγής* (straight exchange sort) βασίζεται στην αρχή της σύγκρισης και ανταλλαγής ζευγών γειτονικών στοιχείων, μέχρις ότου διαταχθούν όλα τα στοιχεία. Σύμφωνα με τη μέθοδο αυτή κάθε φορά γίνονται διαδοχικές προσπελάσεις στον πίνακα και μετακινείται το μικρότερο κλειδί της ακολουθίας προς το αριστερό άκρο του πίνακα. Αν ο πίνακας θεωρηθεί σε κατακόρυφη θέση αντί σε οριζόντια και οι ακέραιοι θεωρηθούν - επιστρατεύοντας αρκετή φαντασία - ως φυσαλίδες (bubbles) σε μία δεξαμενή νερού με βάρη σύμφωνα με την τιμή τους, τότε κάθε προσπέλαση στον πίνακα έχει ως αποτέλεσμα την άνοδο της φυσα-

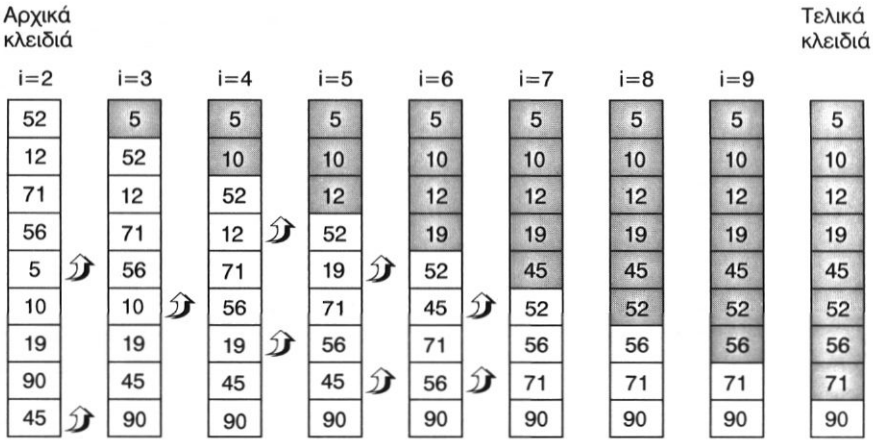
Δομές Δεδομένων δευτερεύουσας μνήμης

Σε μεγάλες πρακτικές εμπορικές/επιστημονικές εφαρμογές, το μέγεθος της κύριας μνήμης δεν επαρκεί για την αποθήκευση των δεδομένων. Στην περίπτωση αυτή χρησιμοποιούνται ειδικές δομές για την αποθήκευση των δεδομένων στη δευτερεύουσα μνήμη, δηλαδή κυρίως στο μαγνητικό δίσκο. Οι ειδικές αυτές δομές ονομάζονται *αρχεία* (files). Είναι γνωστό ότι μία σημαντική διαφορά μεταξύ κύριας μνήμης και μαγνητικού δίσκου είναι ότι στην περίπτωση του δίσκου, τα δεδομένα δεν χάνονται, αν διακοπεί η ηλεκτρική παροχή. Έτσι, τα δεδομένα των αρχείων διατηρούνται ακόμη και μετά τον τερματισμό ενός προγράμματος, κάτι που δεν συμβαίνει στην περίπτωση των δομών της κύριας μνήμης, όπως είναι οι πίνακες, όπου τα δεδομένα χάνονται όταν τελειώσει το πρόγραμμα.Τα στοιχεία ενός αρχείου ονομάζονται *εγγραφές* (records), όπου κάθε εγγραφή αποτελείται από ένα ή περισσότερα *πεδία* (fields), που ταυτοποιούν την εγγραφή, και από άλλα πεδία που περιγράφουν διάφορα χαρακτηριστικά της εγγραφής. Για παράδειγμα, έστω η εγγραφή ενός μαθητή με πεδία: Αριθμός Μητρώου, Ονοματεπώνυμο, Έτος Γέννησης, Τάξη, Τμήμα. Το πεδίο Αριθμός Μητρώου ταυτοποιεί την εγγραφή και ονομάζεται *πρωτεύον κλειδί* (primary key) ή απλά κλειδί. Το πεδίο Ονοματεπώνυμο επίσης ταυτοποιεί την εγγραφή και γι' αυτό αποκαλείται *δευτερεύον κλειδί* (secondary keys), αν υπάρχει πρωτεύον κλειδί. Το πρόβλημα της *αναζήτησης* (searching) μίας εγγραφής με βάση την τιμή του πρωτεύοντος ή ενός δευτερεύοντος κλειδιού σε αρχεία είναι ιδιαίτερα ενδιαφέρον, αν ληφθεί υπ' όψη η μεγάλη ποικιλία των χαρακτηριστικών τόσο της δομής (για παράδειγμα, στατική ή δυναμική, τρόπος οργάνωσης, μέσο αποθήκευσης κ.λπ.), του τύπου των δεδομένων (για παράδειγμα, ακέραιοι, κείμενο, χαρτογραφικά δεδομένα, χρονοσειρές κ.λπ.), όσο και της αναζήτησης (δηλαδή, με βάση το πρωτεύον ή το δευτερεύον κλειδί κλπ.).

λίδας στο κατάλληλο επίπεδο βάρους. Η μέθοδος είναι γνωστή ως ταξινόμηση φυσαλίδας (bubblesort).

Παράδειγμα. Έστω ότι ο αρχικός πίνακας αποτελείται από εννέα κλειδιά τα εξής: 52, 12, 71, 56, 5, 10, 19, 90 και 45. Η μέθοδος εφαρμοζόμενη στε αυτά τα εννέα κλειδιά εξελίσσεται όπως φαίνεται στο επόμενο σχήμα. Κάθε φορά το ταξινομημένο τμήμα του πίνακα εμφανίζεται με χρώμα, ενώ τα στοιχεία που σαν φυσαλίδες ανέρχονται μέσα στον πίνακα εντοπίζο-

νται με το αντίστοιχο βέλος στα δεξιά τους. Κάθε φορά εμφανίζεται η τάξη της επανάληψης (i).



Σχ. 3.7. *Ταξινόμηση φυσαλίδας.*

Η ταξινόμηση φυσαλίδας υλοποιείται με τον επόμενο αλγόριθμο.

```
Αλγόριθμος Φυσαλίδα
Δεδομένα // table, n
Για i από 2 μέχρι n
    Για j από n μέχρι i με_βήμα -1
        Αν table[j-1] > table[j] τότε
            αντιμετάθεσε table[j-1], table[j]
        Τέλος_αν
    Τέλος_επανάληψης
Τέλος_επανάληψης
Αποτελέσματα // table //
Τέλος Φυσαλίδα
```

Για την ταξινόμηση δεδομένων έχουν εκπονηθεί πάρα πολλοί αλγόριθμοι. Άλλοι σχετικά απλοί αλγόριθμοι είναι η ταξινόμηση με επιλογή και η ταξινόμηση με παρεμβολή. Ο πιο γρήγορος αλγόριθμος ταξινόμησης είναι η "γρήγορη ταξινόμηση" (quicksort). Η ταξινόμηση φυσαλίδας είναι ο πιο απλός και ταυτόχρονα ο πιο αργός αλγόριθμος ταξινόμησης.

Στον αλγόριθμο αυτό ως είσοδος δίνεται η μεταβλητή table με n ακεραίους που πρέπει να ταξινομηθούν. Φυσικά η επιλογή του ακέραιου τύπου για το κλειδί είναι αυθαίρετη, αφού μπορεί να χρησιμοποιηθεί οποιοσδήποτε άλλος τύπος, όπου ορίζεται μία συνάρτηση διάταξης, όπως για παράδειγμα ο τύπος του χαρακτήρα.

Σημειώνεται ότι η εντολή "αντιμετάθεσε table[j-1], table[j]" ανταλλάσσει το περιεχόμενο δύο θέσεων με τη βοήθεια μίας βοηθητικής θέσης. Εναλλακτικά αυτό μπορεί να γίνει με τις εξής τρεις εντολές:

```
temp ← table[j-1]
table[j-1] ← table[j]
table[j] ← temp
```

3.8 Αναδρομή

Στο κεφάλαιο αυτό θα εξετάσουμε την έννοια της *αναδρομής* (recursion), που είναι μία σπουδαία εφαρμογή των στοιβών που εξετάσθηκαν προηγουμένως. Η τεχνική της αναδρομής χρησιμοποιείται ευρύτατα τόσο από το λογισμικό συστήματος όσο και στο λογισμικό εφαρμογών. Πιο συγκεκριμένα, η αναδρομή στηρίζεται στη δυνατότητα που προσφέρεται από όλες τις σύγχρονες γλώσσες προγραμματισμού, μία διαδικασία ή συνάρτηση να καλεί τον εαυτό της.

3.8.1 Υπολογισμός του παραγοντικού

Η έννοια του παραγοντικού είναι γνωστή από τα μαθηματικά. Πιο συγκεκριμένα, για ένα ακέραιο n, το n παραγοντικό, που συμβολίζεται με n!, ορίζεται από τη σχέση:

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$$

Ισοδύναμος είναι και ο εξής ορισμός

$$n! = \begin{cases} n \cdot (n-1)! & \text{αν } n > 0 \\ 1 & \text{αν } n = 0 \end{cases}$$

Ο ορισμός αυτός διακρίνεται από το βασικό χαρακτηριστικό ότι το παραγοντικό εκφράζεται με βάση μία απλούστερη περίπτωση του εαυτού του. Δηλαδή εξ ορισμού το παραγοντικό είναι μία αναδρομική αλγεβρική συνάρτηση. Με τον επόμενο αλγόριθμο υπολογίζεται με πολύ απλό αναδρομικό τρόπο το n παραγοντικό.

```
Αλγόριθμος Παραγοντικό
Δεδομένα // n //
Αν n = 0 τότε
    product ← 1
αλλιώς
    product ← n * Παραγοντικό(n-1)
Τέλος_αν
Αποτελέσματα // product //
Τέλος Παραγοντικό
```

Ωστόσο, το n! μπορεί να υπολογισθεί και με επαναληπτική μέθοδο, όπως παρουσιάζεται στον επόμενο αλγόριθμο Παραγοντικό2.

```
Αλγόριθμος Παραγοντικό2
Δεδομένα  // n //
product ← 1
Για i από 2 μέχρι n
    product ← product * i
Τέλος_επανάληψης
Αποτελέσματα  // product //
Τέλος Παραγοντικό2
```

Επομένως, δοθέντων αυτών των δύο λύσεων στο πρόβλημα υπολογισμού του n παραγοντικού προκύπτει το εύλογο ερώτημα, ποια μέθοδος είναι καλύτερη. Αφού εξετάσουμε μερικά ακόμη παραδείγματα, στη συνέχεια θα δοθεί απάντηση στην ερώτηση αυτή.

3.8.2 Υπολογισμός του μέγιστου κοινού διαιρέτη

Ένα ιστορικό πρόβλημα είναι η εύρεση του μέγιστου κοινού διαιρέτη (μκδ) δύο ακεραίων αριθμών. Ο αλγόριθμος εύρεσης του μκδ ανήκει στον Ευκλείδη. Ωστόσο η υλοποίηση και αυτού του αλγορίθμου μπορεί να γίνει κατά πολλούς τρόπους. Στη συνέχεια δίνεται ένας πρώτος αλγόριθμος για τον υπολογισμό του μκδ δύο ακεραίων x και y. Η μέθοδος αυτή είναι αρκετά αργή (και δεν στηρίζεται στον αλγόριθμο του Ευκλείδη), αλλά απλώς δίνεται για να διαφανεί η βελτίωση στην επίδοση από τους επόμενους αλγορίθμους. Ουσιαστικά λαμβάνει το μικρότερο από τους δύο ακεραίους, τον z, και εξετάζει με τη σειρά όλους τους ακεραίους ξεκινώντας από τον z και μειώνοντας συνεχώς κατά μία μονάδα μέχρι και οι δύο αριθμοί, x και y, να διαιρούνται από τη νέα τιμή του z.

```
Αλγόριθμος Μέγιστος_Κοινός_Διαιρέτης
Δεδομένα // x,y //
Αν x < y τότε
    z ← x
αλλιώς
    z ← y
Τέλος_αν
Όσο (x mod z ≠ 0) ή (y mod z ≠ 0) επανάλαβε
    z ← z-1
Τέλος_επανάληψης
Αποτελέσματα // z //
Τέλος Μέγιστος_Κοινός_Διαιρέτης
```

Η επόμενη μέθοδος για την εύρεση του μκδ αποδίδεται στον Ευκλείδη, και προφανώς βελτιώνει την προηγούμενη εκδοχή, επειδή δεν εξετάζει με τη σειρά όλους τους ακεραίους.

```
Αλγόριθμος Ευκλείδης
Δεδομένα  // x,y //
z ← y
Όσο z ≠ 0 επανάλαβε
    z ← x mod y
    x ← y
    y ← z
Τέλος_επανάληψης
Αποτελέσματα //  x //
Τέλος Ευκλείδης
```

Παράδειγμα. Για καλύτερη κατανόηση της μεθόδου, ας παρακολουθήσουμε πως ο αλγόριθμος αυτός βρίσκει το μκδ των αριθμών 150 και 35. Ο επόμενος πίνακας δείχνει τις τιμές των μεταβλητών z, x και y, κατά τη διάρκεια των επαναλήψεων. Δηλαδή, πριν την έναρξη της επαναληπτικής δομής οι αρχικές τιμές των x και y είναι 150 και 35 (όπως φαίνεται στη δεύτερη γραμμή του πίνακα). Ο αλγόριθμος σταματά όταν γίνει 0 η τιμή του z, οπότε ο μκδ είναι η τιμή του x, δηλαδή το 5.

z	x	y
35	150	35
10	35	10
5	10	5
0	5	0

Ωστόσο η μέθοδος του Ευκλείδη μπορεί να υλοποιηθεί και με έναν εναλλακτικό αναδρομικό τρόπο, που δίνεται στη συνέχεια. Η τρίτη αυτή εκδοχή είναι πολύ απλή στον προγραμματισμό και την κατανόησή της.

```
Αλγόριθμος Ευκλείδης
Δεδομένα // x,y //
Αν y = 0 τότε
     z ← x
αλλιώς
    z ← Ευκλείδης(y, x mod y)
Τέλος_αν
Αποτελέσματα // z //
Τέλος Ευκλείδης
```

3.8.3 Υπολογισμός αριθμών ακολουθίας Fibonacci

Για καλύτερη κατανόηση της διαφοράς μεταξύ επαναληπτικών και αναδρομικών μεθόδων, στη συνέχεια θα εξετασθεί ένα τελευταίο παράδειγμα, όπου υπολογίζεται η ακολουθία αριθμών Fibonacci πρώτης τάξης, που ορίζεται ως εξής:

$$F_i = \begin{cases} 0 & \text{αν } i = 0 \\ 1 & \text{αν } i = 1 \\ F_i = F_{i-1} + F_{i-2} & \text{αν } i > 1 \end{cases}$$

ενώ οι πρώτοι όροι της ακολουθίας Fibonacci πρώτης τάξης είναι:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 κλπ.

Από τον ορισμό φαίνεται ανάγλυφα η αναδρομική φύση της συνάρτησης. Οι δύο επόμενοι αλγόριθμοι υπολογίζουν τον αριθμό Fibonacci πρώτης τάξης F_n με επαναληπτική και αναδρομική μέθοδο. Υποτίθεται ότι κατά την κλήση του αλγορίθμου μία μη αρνητική τιμή περνά ως όρισμα στη μεταβλητή n.

```
Αλγόριθμος Fibonacci1
Δεδομένα   // n //
Αν n ≤ 1 τότε Fib ← n
f0 ← 0
f1 ← 1
Για i από 2 μέχρι n
    fib ← f0+f1
    f0 ← f1
    f1 ← fib
Τέλος_επανάληψης
Αποτελέσματα // Fib //
Τέλος Fibonacci1

Αλγόριθμος Fibonacci2
Δεδομένα   // n //
Αν n ≤ 1 τότε
    Fib ← n
αλλιώς
    Fib ← Fibonacci2(n-1) + Fibonacci2(n-2)
Τέλος_Αν
Αποτελέσματα // Fib //
Τέλος Fibonacci2
```

Συχνά η χρήση αναδρομής διευκολύνει τον προγραμματιστή στην υλοποίηση και τον έλεγχο του προγράμματος. Αν και πολλές φορές η αναδρομή φαίνεται ως πιο φυσικός τρόπος προγραμματισμού, ωστόσο πρέπει να χρησιμοποιείται με μέτρο. Μεταξύ ενός απλού επαναληπτικού προγράμματος και ενός αναδρομικού προγράμματος προτιμάται το πρώτο. Ο λόγος είναι ότι κάθε κλήση μίας συνάρτησης ή μίας διαδικασίας έχει χρονικό κόστος μη αμελητέο. Έτσι το κέρδος σε χρόνο προγραμματισμού δημιουργεί απώλεια σε χρόνο εκτέλεσης. Επομένως, αν ο χρόνος απόκρισης είναι κρίσιμος για την εφαρμογή μας, τότε είναι βέβαιο ότι πρέπει να προτιμηθεί η επαναληπτική μέθοδος.

Οι προηγούμενοι αλγόριθμοι για τον υπολογισμό των αριθμών Fibonacci πρώτης τάξης εκτός της διαφοράς τους στην επίδοση λόγω του ότι, η πρώτη είναι επαναληπτική ενώ η δεύτερη είναι αναδρομική, έχουν και μία ακόμη σημαντική διαφορά. Η διαφορά αυτή έγκειται στο γεγονός ότι, η δεύτερη καλεί περισσότερο από μία φορά τον εαυτό της για τις ίδιες τιμές. Αυτό θα γίνει κατανοητό δοκιμάζοντας για παράδειγμα τον υπολογισμό του F_5. Σε επόμενο κεφάλαιο θα επανέλθουμε στη μέλετη των αλγορίθμων αυτών για τον υπολογισμό των αριθμών Fibonacci πρώτης τάξης.

3.9 Άλλες δομές δεδομένων

Κοινό γνώρισμα των δομών που εξετάσθηκαν προηγουμένως είναι ότι οι διαδοχικοί κόμβοι αποθηκεύονται σε συνεχόμενες θέσεις της κύριας μνήμης. Στην παράγραφο αυτή γίνεται μια παρουσίαση τριών πολύ σπουδαίων δομών δεδομένων, στις οποίες οι κόμβοι δεν είναι απαραίτητο να κατέχουν συνεχόμενες θέσεις μνήμης. Πρόκειται για τις *λίστες*, τα *δένδρα* και τους *γράφους*.

3.9.1 Λίστες

Στις λίστες το κύριο χαρακτηριστικό είναι ότι οι κόμβοι τους συνήθως βρίσκονται σε απομακρυσμένες θέσεις μνήμης και η σύνδεσή τους γίνεται με δείκτες. Ο *δείκτης* (pointer) είναι ένας ιδιαίτερος τύπος που προσφέρεται από τις περισσότερες σύγχρονες γλώσσες προγραμματισμού. Ο δείκτης δεν λαμβάνει αριθμητικές τιμές όπως ακέραιες, πραγματικές κ.α., αλλά οι τιμές του είναι διευθύνσεις στην κύρια μνήμη και χρησιμοποιείται ακριβώς για τη σύνδεση των διαφόρων στοιχείων μιας δομής, που είναι αποθηκευμένα σε μη συνεχόμενες θέσεις μνήμης. Συνήθως ο δείκτης είναι ένα πεδίο κάθε κόμβου της δομής, όπως φαίνεται στο σχήμα 3.8. Το πεδίο Δεδομέ-

Δεδομένα	Δείκτης

***Σχ. 3.8** Δομή κόμβου λίστας*

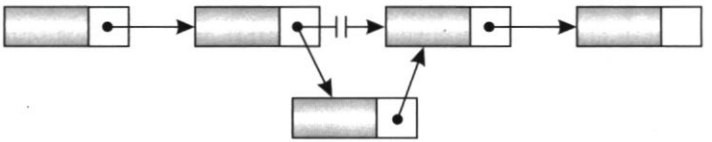
Οι όροι index και pointer αποδίδονται στα ελληνικά ως δείκτης. Και οι δύο παραπέμπουν σε θέσεις, πίνακα ο πρώτος και μνήμης ο δεύτερος.

να μπορεί να περιέχει μία ή περισσότερες αλφαριθμητικές ή αριθμητικές πληροφορίες.

Στο σχήμα 3.9 παρουσιάζεται μια λίστα με τέσσερις κόμβους, όπου οι δείκτες έχουν τη μορφή βέλους, προκειμένου να φαίνεται ο κόμβος που παραπέμπουν.

***Σχ. 3.9.** Μία λίστα με τέσσερις κόμβους*

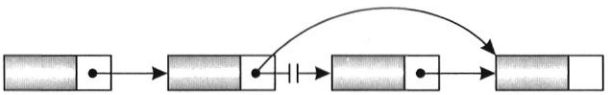
Με τη χρήση δεικτών διευκολύνονται οι λειτουργίες της εισαγωγής και της διαγραφής δεδομένων στις λίστες. Στο σχήμα 3.10 φαίνεται η εισαγωγή ενός νέου κόμβου μεταξύ του δεύτερου και τρίτου κόμβου της προηγούμενης λίστας.



***Σχ. 3.10.** Εισαγωγή σε λίστα*

Οι δομές δεδομένων που χρησιμοποιούν δείκτες, αποκαλούνται δυναμικές (dynamic), γιατί η υλοποίησή τους γίνεται έτσι, ώστε να μην απαιτείται εκ των προτέρων καθορισμός του μέγιστου αριθμού κόμβων. Είναι φανερό, ότι οι δομές αυτές είναι πιο ευέλικτες από τη στατική δομή του πίνακα, επειδή επεκτείνονται και συρρικνώνονται κατά τη διάρκεια εκτέλεσης του προγράμματος.

Οπως φαίνεται και στο σχήμα, οι απαιτούμενες ενέργειες για την εισαγωγή (παρεμβολή) του νέου κόμβου είναι ο δείκτης του δεύτερου κόμβου να δείχνει το νέο κόμβο και ο δείκτης του νέου κόμβου να δείχνει τον τρίτο κόμβο (δηλαδή να πάρει την τιμή που είχε πριν την εισαγωγή ο δείκτης του δεύτερου κόμβου). Ετσι οι κόμβοι της λίστας διατηρούν τη λογική τους σειρά, αλλά οι φυσικές θέσεις στη μνήμη μπορεί να είναι τελείως διαφορετικές.

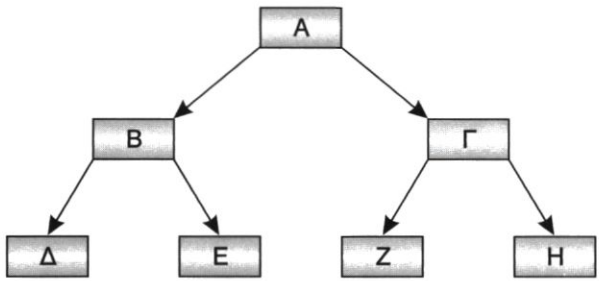


***Σχ. 3.11.** Διαγραφή κόμβου λίστας*

Αντίστοιχα για τη διαγραφή ενός κόμβου αρκεί ν' αλλάξει τιμή ο δείκτης του προηγούμενου κόμβου και να δείχνει πλέον τον επόμενου αυτού που διαγράφεται, όπως φαίνεται στο σχήμα 3.11. Ο κόμβος που διαγράφηκε (ο τρίτος) αποτελεί "άχρηστο δεδομένο" και ο χώρος μνήμης που καταλάμβανε, παραχωρείται για άλλη χρήση.

3.9.2 Δένδρα

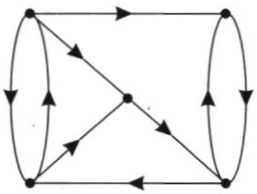
Τα *δένδρα* (trees) είναι δομές που στις σύγχρονες γλώσσες προγραμματισμού υλοποιούνται με τη βοήθεια των δεικτών, όπως εξηγήθηκε στην αρχή αυτής της παραγράφου. Βέβαια, μπορούν να υλοποιηθούν και με στατικές δομές (με πίνακες). Το κύριο χαρακτηριστικό των δένδρων είναι, ότι από ένα κόμβο δεν υπάρχει ένας μόνο επόμενος κόμβος, αλλά περισσότεροι. Υπάρχει ένας μόνο κόμβος, που λέγεται *ρίζα*, από τον οποίο ξεκινούν όλοι οι άλλοι κόμβοι. Στο σχήμα 3.12. παρατηρούμε ότι από τη ρίζα ξεκινούν δύο κόμβοι. Οι κόμβοι αυτοί λέγονται *παιδιά* της ρίζας. Με την ίδια λογική, από κάθε παιδί της ρίζας ξεκινούν άλλα παιδιά κ.ο.κ. Στη βιβλιογραφία αναφέρεται μία τεράστια ποικιλία δομών δένδρων, που η αναφορά σε αυτές βρίσκεται εκτός των ορίων του βιβλίου αυτού.



Σχ. 3.12. *Δομή δένδρου*

3.9.3 Γράφοι

Ενας γράφος (graph) αποτελείται από ένα σύνολο κόμβων (ή σημείων ή κορυφών) και ένα σύνολο γραμμών (ή ακμών ή τόξων) που ενώνουν μερικούς ή όλους τους κόμβους. Ο γράφος αποτελεί την πιο γενική δομή δεδομένων, με την έννοια ότι όλες οι προηγούμενες δομές που παρουσιάστηκαν μπορούν να θεωρηθούν περιπτώσεις γράφων.



Σχ. 3.13. *Ένας γράφος*

Πολλά προβλήματα και καταστάσεις της καθημερινής μας ζωής μπορούν να περιγραφούν με τη βοήθεια γράφων. Για παράδειγμα τα σημεία ενός γράφου μπορούν να παριστούν πόλεις και οι γραμμές τις οδικές συνδέσεις μεταξύ τους. Λόγω της μεγάλης πληθώρας και ποικιλίας των προβλημάτων που σχετίζονται με γράφους, έχει αναπτυχθεί ομώνυμη θεωρία, η Θεωρία Γράφων, η οποίο συχνά αποτελεί αυτοδύναμο μάθημα σε πανεπιστημιακά τμήματα.

Ανακεφαλαίωση

Στο κεφάλαιο αυτό αρχικά ορίσθηκε η Πληροφορική ως η επιστήμη που μελετά τα δεδομένα από τις σκοπιές του υλικού, των γλωσσών προγραμματισμού, των δομών δεδομένων και της ανάλυσης δεδομένων. Δόθηκε ο ορισμός της δομής δεδομένων και ένας κατάλογος με τις λειτουργίες που μπορούν να γίνουν με μία δομή δεδομένων. Η πρώτη δομή που εξετάσθηκε ήταν η δομή του πίνακα (μονοδιάστατου και δισδιάστατου), που είναι μία στατική δομή, με μέγεθος που δεν μεταβάλλεται χρονικά. Στη συνέχεια παρουσιάσθηκε η δομή της στοίβας, καθώς και των δύο βασικών πράξεων της ώθησης και της απώθησης των στοιχείων της. Επίσης περιγράφηκε η δομή της ουράς με αναφορά στις πράξεις της εισαγωγής και της εξαγωγής στοιχείων από αυτήν. Στη συνέχεια παρουσιάσθηκαν προβλήματα η λύση των οποίων εντάσσεται στις κατηγορίες της αναζήτησης και της ταξινόμησης. Η τεχνική της σειριακής/γραμμικής αναζήτησης στοιχείων από πίνακα δόθηκε με χρήση σχετικών αλγορίθμων και προσδιορίσθηκαν οι περιπτώσεις όπου η μέθοδος αυτή είναι αποτελεσματική. Έγινε σημαντική εμβάθυνση στη μέθοδο της αναδρομής μέσω διαφόρων παραδειγμάτων επίλυσης γνωστών προβλημάτων. Τέλος, επιγραμματικά παρουσιάζονται οι δομές της λίστας, του δένδρου και του γράφου.

Λέξεις κλειδιά

Δεδομένα, Πληροφορία, Δομή δεδομένων, Στατικές και δυναμικές δομές, Πίνακες, Στοίβα, Ουρά, FIFO και LIFO, Γραμμική αναζήτηση, Ταξινόμηση, Αναδρομή, Λίστες, Δένδρα, Γράφοι.

Ερωτήσεις - Θέματα για συζήτηση

1. Τι είναι δεδομένα και τι είναι πληροφορία ; Να δοθεί σύντομος ορισμός των όρων αυτών.
2. Ποιές είναι οι απόψεις από τις οποίες η επιστήμη της Πληροφορικής μελετά τα δεδομένα;
3. Να δοθεί ο ορισμός της δομής δεδομένων.
4. Ποιές είναι οι βασικές πράξεις επί των δομών δεδομένων;
5. Ποιά είναι η εξάρτηση μεταξύ της δομής δεδομένων και του αλγορίθμου που επεξεργάζεται τη δομή;
6. Να περιγραφούν οι δύο κυριότερες κατηγορίες των δομών δεδομένων.
7. Να περιγραφεί η δομή του πίνακα και να δοθεί παράδειγμα χρήσης του.
8. Να δοθεί ο ορισμός της στοίβας.
9. Ποιές είναι οι βασικές λειτουργίες που γίνονται σε μία στοίβα;
10. Να δοθεί ο ορισμός της ουράς.
11. Ποιές είναι οι βασικές λειτουργίες που γίνονται σε μία ουρά;
12. Να περιγραφεί η λειτουργία της αναδρομής και να σχολιασθεί η χρησιμότητα της.
13. Να δοθεί αναδρομικός αλγόριθμος υπολογισμού της δύναμης παραγματικού αριθμού υψωμένου σε ακέραια δύναμη.
14. Να περιγραφεί η λειτουργία της αναζήτησης.
15. Να δοθεί ένα παράδειγμα για τη σειριακή αναζήτηση στοιχείου σε έναν πίνακα.
16. Να δοθεί ο ορισμός της έννοιας της ταξινόμησης.
17. Να περιγραφεί η ταξινόμηση ευθείας ανταλλαγής και να δοθεί ένα παράδειγμα.

Βιβλιογραφία

1. Νικόλαος Γλυνός, *Δομές Δεδομένων, Πανεπιστήμιο Ιωαννίνων*, 1996.
2. Χρήστος Κοίλιας, *Δομές Δεδομένων και Οργάνωση Αρχείων*. Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1993.

3. Ιωάννης Μανωλόπουλος, *Δομές Δεδομένων – μία Προσέγγιση με Pascal*, Εκδόσεις Art of Text, Θεσσαλονίκη, 1998.

4. Νικόλαος Μισυρλής, *Δομές Δεδομένων*, Αθήνα, 1993.

5. D. Brunskill and J. Turner: *"Understanding Algorithms and Data Structures"*, McGraw-Hill, 1996.

6. D. E. Knuth: *"The Art of Computer Programming : Fundamental Algorithms"*, Vol.1, 3rd edition, Addison Wesley, 1997.

7. M.A. Weiss: "Data Structures and Algorithm Analysis", 2nd edition, Benjamin/Cummings, 1995.

Διευθύνσεις Διαδικτύου

http://www.nist.gov/dads/

Κόμβος με ευρετήριο όρων για αλγορίθμους, Δομές Δεδομένων και Προβλήματα (Algorithms, Data Structures, and Problems Terms and Definitions for the CRC Dictionary of Computer Science, Engineering and Technology).

ΚΕΦΑΛΑΙΟ 10ο

ΥΠΟΠΡΟΓΡΑΜΜΑΤΑ

ΠΕΡΙΕΧΟΜΕΝΑ

- Τμηματικός προγραμματισμός
- Χαρακτηριστικά των υποπρογραμμάτων
- Πλεονεκτήματα του τμηματικού προγραμματισμού
- Παράμετροι
- Διαδικασίες και συναρτήσεις
- Εμβέλεια μεταβλητών - σταθερών
- Αναδρομή

Εισαγωγή

Η επίλυση ενός προβλήματος διευκολύνεται με τη διαίρεση του σε μικρότερα υποπροβλήματα. Η επίλυση των υποπροβλημάτων αυτών οδηγεί στην επίλυση του αρχικού προβλήματος. Ο τμηματικός προγραμματισμός, η διαίρεση δηλαδή ενός προγράμματος σε υποπρογράμματα υλοποιεί αυτήν την ιδέα στον προγραμματισμό. Το κεφάλαιο αυτό ασχολείται με τις αρχές του τμηματικού προγραμματισμού, τα είδη των υποπρογραμμάτων που υποστηρίζει η **ΓΛΩΣΣΑ**, τις διαδικασίες και τις συναρτήσεις καθώς και τον τρόπο που τα υποπρογράμματα αυτά επικοινωνούν μεταξύ τους. Τέλος παρουσιάζεται και αναλύεται ο τρόπος υλοποίησης αναδρομικών αλγορίθμων με χρήση αναδρομικών υποπρογραμμάτων.

Στόχοι

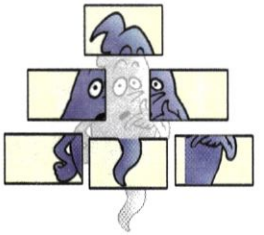
Να είναι σε θέση ο μαθητής :

- Να αναλύει ένα σύνθετο πρόγραμμα σε απλά υποπρογράμματα.
- Να διακρίνει τις συναρτήσεις από τις διαδικασίες και να επιλέγει τη χρήση διαδικασίας ή συνάρτησης για την υλοποίηση ενός υποπρογράμματος.
- Να περιγράφει τη δομή των υποπρογραμμάτων και να χρησιμοποιεί παραμέτρους για την επικοινωνία τους.
- Να καθορίζει τις περιοχές εμβέλειας των παραμέτρων.
- Να συντάσει αναδρομικά υποπρογράμματα και να συγκρίνει αναδρομικές και επαναληπτικές διαδικασίες.

Προερωτήσεις

- ✓ Η δόμηση ενός προγράμματος με τη μορφή ενός συνόλου μικρότερων προγραμμάτων βοηθάει τον προγραμματιστή στην ανάπτυξη ενός σύνθετου προγράμματος;
- ✓ Νομίζετε ότι όλα τα είδη τμημάτων προγραμμάτων επιτελούν την ίδια εργασία;
- ✓ Πως μπορεί να οργανώνεται ένα πρόγραμμα σε μικρότερα προγράμματα;
- ✓ Χρειάζεται τα επιμέρους προγράμματα να επικοινωνούν μεταξύ τους;
- ✓ Γνωρίζεις από την άλγεβρα τους αναδρομικούς τύπους; Ποια τα πλεονεκτήματα τους;

10.1. Τμηματικός προγραμματισμός



Τα προβλήματα που αντιμετωπίστηκαν στα προηγούμενα κεφάλαια, ήταν αρκετά απλά, ώστε να μπορούν να αναπτυχθούν σωστά σε ένα και μόνο πρόγραμμα. Όπως αναφέρθηκε στο κεφάλαιο 6, ο καλύτερος τρόπος για να αντιμετωπισθούν σύνθετα προβλήματα και να γραφούν τα αντίστοιχα προγράμματα, είναι η ιεραρχική προσέγγιση, η ανάπτυξη του προγράμματος από επάνω προς τα κάτω (top-down). Κάθε πρόβλημα διαιρείται σε μικρότερα επιμέρους προβλήματα και κάθε ένα από αυτά τα προγράμματα διαιρείται σε ακόμα απλούστερα και μικρότερα. Στο τέλος τα επί μέρους υπο-προβλήματα είναι αρκετά απλά, ώστε οι αντίστοιχοι αλγόριθμοι και τα αντίστοιχα τμήματα προγράμματος να μπορούν να σχεδιασθούν και να γραφούν εύκολα. Ο τελικός αλγόριθμος του προβλήματος ανάγεται σε πολλούς απλούστερους επί μέρους αλγόριθμους και το τελικό πρόγραμμα σε πολλά απλούστερα τμήματα προγράμματος.

Η τεχνική του τμηματικού προγραμματισμού είναι ένα από τα βασικότερα συστατικά του δομημένου προγραμματισμού, ο οποίος εξασφαλίζει σε μεγάλο βαθμό την επιτυχή και εύκολη δημιουργία σωστών προγραμμάτων.

Τμηματικός προγραμματισμός ονομάζεται η τεχνική σχεδίασης και ανάπτυξης των προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων.

Παράδειγμα 1

Ας μελετήσουμε κατ' αρχήν το πρόβλημα που μας απασχόλησε στο πρώτο κεφάλαιο του βιβλίου, την αξιολόγηση των αποτελεσμάτων των μαθητών Γ' Λυκείου στα μαθήματα ειδικότητας.

Το σύνθετο αυτό πρόβλημα για να αντιμετωπισθεί πιο εύκολα πρέπει να αναλυθεί σε επιμέρους μικρότερα προβλήματα.

Συγκεκριμένα τα τρία βασικά διαφορετικά τμήματα είναι:

- Εισαγωγή δεδομένων
- Επεξεργασία δεδομένων
- Εκτύπωση αποτελεσμάτων

Τα τρία αυτά τμήματα μπορούν να αναλυθούν περισσότερο. Συγκεκριμένα :

Εισαγωγή δεδομένων

- Καταχώριση δεδομένων
- Έλεγχος δεδομένων

Επεξεργασία δεδομένων

- Υπολογισμός μέσης τιμής
- Υπολογισμός τυπικής απόκλισης
- Κατανομή συχνοτήτων
- Δημιουργία γραφικών παραστάσεων

Εκτύπωση αποτελεσμάτων

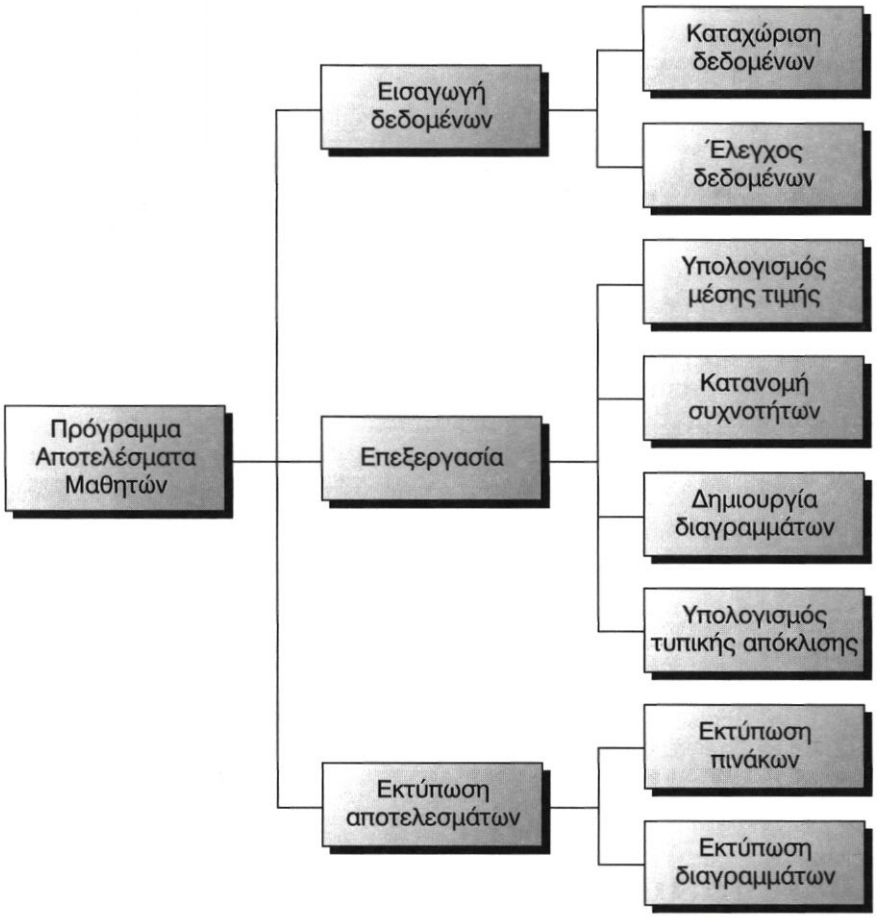
- Εκτύπωση πινάκων συχνοτήτων
- Εκτύπωση γραφικών παραστάσεων

Οπως φαίνεται το αρχικό πρόβλημα διασπάστηκε σε αρκετά απλούστερα υποπροβλήματα. Η δημιουργία λοιπόν του τελικού προγράμματος ανάγεται στη δημιουργία των επί μέρους τμημάτων προγραμμάτων ή ενοτήτων και τη σύνδεση αυτών μεταξύ τους. Μερικά από αυτά τα τμήματα, όπως ο υπολογισμός της μέσης τιμής ή της τυπικής απόκλισης, έχουν ήδη αντιμετωπιστεί στο προηγούμενο κεφάλαιο, που σημαίνει ότι μπορούμε να εκμεταλλευτούμε τα προγράμματα που ήδη έχουμε γράψει μειώνοντας έτσι την εργασία για την επίλυση του προβλήματος.

Η παράσταση της ανάλυσης του προβλήματος μπορεί να γίνει γραφικά με το διάγραμμα του σχήματος 10.1

Η έννοια του τμηματικού προγραμματισμού έχει ήδη αποτυπωθεί και σε προηγούμενα κεφάλαια. Για παράδειγμα στο πρόγραμμα υπολογισμού των θερμοκρασιών του προηγούμενου κεφαλαίου (παράδειγμα 3) το τμήμα της εισαγωγής δεδομένων έχει ξεχωρίσει από το τμήμα υπολογισμών σε αντίθεση με τα παραδείγματα 1 και 2 που είναι ενιαία.

Οταν ένα τμήμα προγράμματος επιτελεί ένα αυτόνομο έργο και έχει γραφεί χωριστά από το υπόλοιπο πρόγραμμα, τότε αναφερόμαστε σε **υποπρόγραμμα** (subprogram).



Σχ. 10.1 Διαγραμματική απεικόνιση της ανάλυσης προβλήματος

10.2. Χαρακτηριστικά των υποπρογραμμάτων

Ο χωρισμός ενός προγράμματος σε υποπρογράμματα προυποθέτει την ανάλυση του αρχικού προβλήματος σε μικρότερα υποπροβλήματα, τα οποία να μπορούν να αντιμετωπισθούν ανεξάρτητα το ένα από το άλλο. Η ανάλυση όμως αυτή δεν είναι πάντα εύκολη και όπως ισχύει γενικά στον προγραμματισμό, δεν υπάρχουν συγκεκριμένοι κανόνες για την επιτυχή ανάλυση. Η δυσκολία δε αυξάνεται όσο πιο μεγάλο και πιο σύνθετο είναι το πρόβλημα. Η σωστή εφαρμογή του τμηματικού προγραμματισμού απαιτεί

μελέτη στην ανάλυση του προβλήματος, εμπειρία στον προγραμματισμό, ταλέντο και φυσικά γνώσεις.

Υπάρχουν πάντως τρεις ιδιότητες που πρέπει να διακρίνουν τα υποπρογράμματα:

- **Κάθε υποπρόγραμμα έχει μόνο μία είσοδο και μία έξοδο.** Στην πραγματικότητα κάθε υποπρόγραμμα ενεργοποιείται με την είσοδο σε αυτό που γίνεται πάντοτε από την αρχή του, εκτελεί ορισμένες ενέργειες, και απενεργοποιείται με την έξοδο από αυτό που γίνεται πάντοτε από το τέλος του.
- **Κάθε υποπρόγραμμα πρέπει να είναι ανεξάρτητο από τα άλλα**. Αυτό σημαίνει ότι κάθε υποπρόγραμμα μπορεί να σχεδιαστεί, να αναπτυχθεί και να συντηρηθεί αυτόνομα χωρίς να επηρεαστούν άλλα υποπρογράμματα. Στην πράξη βέβαια η απόλυτη ανεξαρτησία είναι δύσκολο να επιτευχθεί.
- **Κάθε υποπρόγραμμα πρέπει να μην είναι πολύ μεγάλο**. Η έννοια του μεγάλου προγράμματος είναι υποκειμενική, αλλά πρέπει κάθε υποπρόγραμμα να είναι τόσο, ώστε να είναι εύκολα κατανοητό για να μπορεί να ελέγχεται. Γενικά κάθε υποπρόγραμμα πρέπει να εκτελεί μόνο μία λειτουργία. Αν εκτελεί περισσότερες λειτουργίες, τότε συνήθως μπορεί και πρέπει να διασπαστεί σε ακόμη μικρότερα υποπρογράμματα.

10.3. Πλεονεκτήματα του τμηματικού προγραμματισμού

Η χρήση υποπρογραμμάτων σε ένα πρόγραμμα παρουσιάζει πολλά πλεονεκτήματα που αναφέρθηκαν συνοπτικά στο κεφάλαιο 6. Η σωστή χρήση του τμηματικού προγραμματισμού, δηλαδή ο σωστός χωρισμός ενός σύνθετου προγράμματος σε υποπρογράμματα εξασφαλίζει τέσσερα βασικά χαρακτηριστικά του σωστού προγραμματισμού:

Διευκολύνει την ανάπτυξη του αλγορίθμου και του αντιστοίχου προγράμματος.

Επιτρέπει την εξέταση και την επίλυση απλών προβλημάτων και όχι στην αντιμετώπιση του συνολικού προβλήματος. Με τη σταδιακή επίλυση των υποπροβλημάτων και τη δημιουργία των αντιστοίχων υποπρογραμμάτων τελικά επιλύεται το συνολικό πρόβλημα.

Διευκολύνει την κατανόηση και διόρθωση του προγράμματος.

Ο χωρισμός του προγράμματος σε μικρότερα αυτοτελή τμήματα επιτρέπει τη γρήγορη διόρθωση ενός συγκεκριμένου τμήματος του χωρίς οι αλλαγές αυτές να επηρεάσουν όλο το υπόλοιπο πρόγραμμα.

Επίσης διευκολύνει οποιονδήποτε χρειαστεί να διαβάσει και να κατανοήσει τον τρόπο που λειτουργεί το πρόγραμμα. Όπως έχει πολλές φορές τονιστεί αυτό είναι πολύ σημαντικό χαρακτηριστικό του σωστού προγραμματισμού, αφού ένα μεγάλο πρόγραμμα στον κύκλο της ζωής του χρειάζεται να συντηρηθεί από διαφορετικούς προγραμματιστές.

Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος.

Πολύ συχνά χρειάζεται η ίδια λειτουργία σε διαφορετικά σημεία ενός προγράμματος. Από τη στιγμή που ένα υποπρόγραμμα έχει γραφεί, μπορεί το ίδιο να καλείται από πολλά σημεία του προγράμματος. Έτσι μειώνονται το μέγεθος του προγράμματος, ο χρόνος που απαιτείται για τη συγγραφή του και οι πιθανότητες λάθους, ενώ ταυτόχρονα το πρόγραμμα γίνεται πιο εύληπτο και κατανοητό.

Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού.

Ένα υποπρόγραμμα που έχει γραφεί μπορεί να χρησιμοποιηθεί πολύ εύκολα και σε άλλα προγράμματα. Από τη στιγμή που έχει δημιουργηθεί, η χρήση του δεν διαφέρει από τη χρήση των ενσωματωμένων συναρτήσεων που παρέχει η γλώσσα προγραμματισμού, όπως για τον υπολογισμό του ημίτονου ή του συνημίτονου ή την εντολή με την οποία εκτελεί μία συγκεκριμένη διαδικασία, για παράδειγμα γράφει στην οθόνη (εντολή ΓΡΑΨΕ). Αν λοιπόν χρειάζεται συχνά κάποια λειτουργία που δεν υποστηρίζεται απευθείας από τη γλώσσα, όπως για παράδειγμα η εύρεση του μικρότερου δύο αριθμών, τότε μπορεί να γραφεί το αντίστοιχο υποπρόγραμμα. Η συγγραφή πολλών υποπρογραμμάτων και η δημιουργία βιβλιοθηκών με αυτά, ουσιαστικά επεκτείνουν την ίδια τη γλώσσα προγραμματισμού.

10.4. Παράμετροι

Τα υποπρογράμματα ενεργοποιούνται από κάποιο άλλο πρόγραμμα ή υποπρόγραμμα για να εκτελέσουν συγκεκριμένες λειτουργίες.

Κάθε υποπρόγραμμα για να ενεργοποιηθεί καλείται, όπως λέγεται, από ένα άλλο υποπρόγραμμα ή το αρχικό πρόγραμμα, το οποίο ονομάζεται *κύ-*

ριο πρόγραμμα. Το υποπρόγραμμα είναι αυτόνομο και ανεξάρτητο τμήμα προγράμματος, αλλά συχνά πρέπει να επικοινωνεί με το υπόλοιπο πρόγραμμα. Συνήθως δέχεται τιμές από το τμήμα προγράμματος που το καλεί και μετά την εκτέλεση επιστρέφει σε αυτό νέες τιμές, αποτελέσματα.

Οι τιμές αυτές που περνούν από το ένα υποπρόγραμμα στο άλλο λέγονται *παράμετροι*.

Οι παράμετροι λοιπόν είναι σαν τις κοινές μεταβλητές ενός προγράμματος με μία ουσιώδη διαφορά, χρησιμοποιούνται για να περνούν τιμές στα υποπρογράμματα.

Μία παράμετρος είναι μία μεταβλητή που επιτρέπει το πέρασμα της τιμής της από ένα τμήμα προγράμματος σε ένα άλλο.

Παραδείγματα παραμέτρων, καθώς και ο τρόπος που περνούν οι τιμές προς και από το υποπρόγραμμα, θα δοθούν στη συνέχεια.

10.5. Διαδικασίες και συναρτήσεις

Υπάρχουν δύο ειδών υποπρογράμματα, οι **διαδικασίες** και οι **συναρτήσεις**. Το είδος κάθε υποπρογράμματος καθορίζεται από το είδος της λειτουργίας που καλείται να επιτελέσει.

Οι διαδικασίες μπορούν να εκτελέσουν οποιαδήποτε λειτουργία από αυτές που μπορεί να εκτελέσει ένα πρόγραμμα. Να εισάγουν δεδομένα, να εκτελέσουν υπολογισμούς, να μεταβάλλουν τις τιμές των μεταβλητών και να τυπώσουν αποτελέσματα. Με τη χρήση των παραμέτρων αυτές τις τιμές μπορούν να τις μεταφέρουν και στα άλλα υποπρογράμματα.

Αντίθετα η λειτουργία των συναρτήσεων είναι πιο περιορισμένη. Οι συναρτήσεις υπολογίζουν μόνο μία τιμή, αριθμητική, χαρακτήρα ή λογική και μόνο αυτήν επιστρέφουν στο υποπρόγραμμα που την κάλεσε. Οι συναρτήσεις μοιάζουν με τις συναρτήσεις των μαθηματικών και η χρήση τους είναι όμοια με τη χρήση των ενσωματωμένων συναρτήσεων που υποστηρίζει η γλώσσα προγραμματισμού.

Ο τρόπος κλήσης καθώς και ο τρόπος σύνταξης των δύο αυτών τύπων των υποπρογραμμάτων είναι διαφορετικός. Τόσο οι συναρτήσεις όσο και οι διαδικασίες τοποθετούνται μετά το τέλος του κυρίου προγράμματος.

Οι συναρτήσεις εκτελούνται απλά με την εμφάνιση του ονόματος τους σε οποιαδήποτε έκφραση, ενώ για να εκτελεστούν οι διαδικασίες χρησιμοποιείται η ειδική εντολή ΚΑΛΕΣΕ και το όνομα της διαδικασίας.

Η **συνάρτηση** είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή με το όνομά της (όπως οι μαθηματικές συναρτήσεις).

Η **διαδικασία** είναι ένας τύπος υποπρογράμματος που μπορεί να εκτελεί όλες τις λειτουργίες ενός προγράμματος.

Ας δούμε ένα απλό παράδειγμα χρήσης διαδικασιών και συναρτήσεων.

Παράδειγμα 2

Να γραφεί πρόγραμμα, το οποίο υπολογίζει το εμβαδό του κύκλου από την ακτίνα του.

Το πρόγραμμα εκτελεί τρεις συγκεκριμένες απλές λειτουργίες.

α) Διαβάζει τα δεδομένα, την ακτίνα η οποία πρέπει να είναι θετικός αριθμός

β) Υπολογίζει το εμβαδό ($E=\pi r^2$)

γ) Τυπώνει το αποτέλεσμα, το εμβαδό, Ε

Αν και το πρόγραμμα είναι πολύ απλό και μπορεί κάλλιστα να γραφεί χωρίς τη χρήση υποπρογραμμάτων, ας το διασπάσουμε σε τρία υποπρογράμματα που εκτελούν τις τρεις παραπάνω λειτουργίες.

Τα πρώτο υποπρόγραμμα πρέπει να διαβάζει την ακτίνα και να την επιστρέφει στο κύριο πρόγραμμα. Αφού το υποπρόγραμμα πρέπει να διαβάζει δεδομένα, υλοποιείται με διαδικασία. Η διαδικασία αυτή που ονομάζεται Είσοδος_δεδομένων, δέχεται από το πληκτρολόγιο την τιμή της ακτίνας που την καταχωρεί στη μεταβλητή Αριθμός και έχει ως εξής:

```
ΔΙΑΔΙΚΑΣΙΑ Είσοδος_δεδομένων(Αριθμός)
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ :  Αριθμός
ΑΡΧΗ
    ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
        ΓΡΑΨΕ 'Δώσε την ακτίνα'
        ΔΙΑΒΑΣΕ Αριθμός
    ΜΕΧΡΙΣ_ΟΤΟΥ Αριθμός>0
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Το δεύτερο πρέπει να υπολογίζει το εμβαδό και να επιστρέφει την τιμή στο κύριο πρόγραμμα. Το υποπρόγραμμα αυτό παίρνει την τιμή της ακτίνας και επιστρέφει μόνο μία τιμή την τιμή του Εμβαδού. Μπορεί λοιπόν να υλοποιηθεί με μία συνάρτηση, η οποία επιστρέφει έναν πραγματικό αριθμό.

Η συνάρτηση Εμβαδό _κύκλου(R) δέχεται έναν πραγματικό αριθμό και υπολογίζει το εμβαδό που επίσης είναι ένας πραγματικός αριθμός. Το είδος της συνάρτησης, δηλαδή η τιμή που επιστρέφει δηλώνεται στην αρχή της συνάρτησης.

```
ΣΥΝΑΡΤΗΣΗ  Εμβαδό_κύκλου(R)  : ΠΡΑΓΜΑΤΙΚΗ
ΣΤΑΘΕΡΕΣ
    Π=3.14
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ: R
ΑΡΧΗ
    Εμβαδό_κύκλου <- Π*R^2
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Το τρίτο υποπρόγραμμα τυπώνει το αποτέλεσμα. Εφόσον απαιτείται από αυτό η εκτέλεση της λειτουργίας της εκτύπωσης, πρέπει να υλοποιηθεί με διαδικασία. Η διαδικασία Εκτύπωση δέχεται από το κύριο πρόγραμμα μια τιμή στη μεταβλητή Αποτέλεσμα και την εκτυπώνει.

```
ΔΙΑΔΙΚΑΣΙΑ Εκτύπωση(Αποτέλεσμα)
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Αποτέλεσμα
ΑΡΧΗ
    ΓΡΑΨΕ 'Το εμβαδό του κύκλου είναι :',Αποτέλεσμα
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Το κύριο πρόγραμμα που καλεί όλα τα υποπρογράμματα έχει ως εξής:

```
ΠΡΟΓΡΑΜΜΑ Παράδειγμα_2
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : R, Εμ
ΑΡΧΗ
    ΚΑΛΕΣΕ Είσοδος_δεδομένων (R)
    Εμ <- Εμβαδό_κύκλου (R)
    ΚΑΛΕΣΕ Εκτύπωση (Εμ)
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Το πρόγραμμα πλέον έχει ολοκληρωθεί. Οταν εκτελεστεί, θα ζητήσει από το χρήστη να εισάγει μια τιμή για την ακτίνα και θα εμφανίσει το εμβα-

δό του κύκλου. Αν η εισαγόμενη τιμή για την ακτίνα είναι 10, τότε θα η οθόνη θα παρουσιάζει τα εξής:

```
Δώσε την ακτίνα
10
Το εμβαδό του κύκλου είναι : 314
```

Ωστόσο υπάρχουν μερικά λεπτά σημεία που αφορούν στο πέρασμα τιμών, τα οποία θα διευκρινιστούν στη συνέχεια.

10.5.1 Ορισμός και κλήση συναρτήσεων

Κάθε συνάρτηση έχει την ακόλουθη δομή.

```
ΣΥΝΑΡΤΗΣΗ όνομα (λίστα παραμέτρων):τύπος συνάρτησης
Τμήμα δηλώσεων
ΑΡΧΗ
    ....
    όνομα <- έκφραση
    ...
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

Το όνομα της συνάρτησης είναι οποιοδήποτε έγκυρο όνομα της **ΓΛΩΣΣΑΣ**. Η λίστα παραμέτρων είναι μια λίστα μεταβλητών, των οποίων οι τιμές μεταβιβάζονται στη συνάρτηση κατά την κλήση.

Οι συναρτήσεις μπορούν να επιστρέφουν τιμές όλων των τύπων δεδομένων που υποστηρίζει η γλώσσα. Μια συνάρτηση λοιπόν μπορεί να είναι ΠΡΑΓΜΑΤΙΚΗ, ΑΚΕΡΑΙΑ, ΧΑΡΑΚΤΗΡΑΣ, ΛΟΓΙΚΗ

Στις εντολές του σώματος της συνάρτησης πρέπει υποχρεωτικά να υπάρχει μία εντολή εκχώρησης τιμής στο όνομα της συνάρτησης, στο προηγούμενο παράδειγμα Εμβαδό_κύκλου<-Π*R^2.

Κάθε συνάρτηση εκτελείται, όπως ακριβώς εκτελούνται οι ενσωματωμένες συναρτήσεις της γλώσσας. Απλώς αναφέρεται το όνομα της σε μια έκφραση ή σε μία εντολή και επιστρέφεται η τιμή της. Στο παράδειγμα η συνάρτηση εκτελείται με την εντολή Εμ<-Εμβαδό_κύκλου(R).

Ο μηχανισμός που επιτυγχάνεται αυτό, είναι ο εξής: Το κύριο πρόγραμμα πριν την κλήση της συνάρτησης γνωρίζει την τιμή της μεταβλητής R. Κατά την κλήση μεταβιβάζεται αυτή η τιμή στην αντίστοιχη μεταβλητή R της συνάρτησης. Η συνάρτηση υπολογίζει το εμβαδό του κύκλου και το αποτέλεσμα αυτό εκχωρείται στο όνομα της συνάρτησης. Με το τέλος της συνάρτησης γίνεται επιστροφή στο κύριο πρόγραμμα, όπου η τιμή του εμβαδού εκχωρείται στη μεταβλητή Εμ.

10.5.2 Ορισμός και κλήση διαδικασιών

Κάθε διαδικασία έχει την ακόλουθη δομή.

```
ΔΙΑΔΙΚΑΣΙΑ Όνομα (λίστα παραμέτρων)
Τμήμα δηλώσεων
ΑΡΧΗ
    εντολές
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Το όνομα της διαδικασίας είναι οποιοδήποτε έγκυρο όνομα της **ΓΛΩΣΣΑΣ**. Η λίστα παραμέτρων είναι μια λίστα μεταβλητών, των οποίων οι τιμές μεταβιβάζονται προς τη διαδικασία κατά την κλήση ή/και επιστρέφονται στο κύριο πρόγραμμα μετά το τέλος της διαδικασίας. Στο σώμα της διαδικασίας μπορούν να υπάρχουν οποιεσδήποτε εντολές της γλώσσας.

Κάθε διαδικασία εκτελείται όταν καλείται από το κύριο πρόγραμμα ή άλλη διαδικασία. Η κλήση σε διαδικασία πραγματοποιείται με την εντολή **ΚΑΛΕΣΕ**, που ακολουθείται από το όνομα της διαδικασίας συνοδευόμενο μέσα σε παρενθέσεις με τη λίστα παραμέτρων.

Η γενική μορφή της εντολής **ΚΑΛΕΣΕ** είναι

Σύνταξη

```
ΚΑΛΕΣΕ όνομα-διαδικασίας (λίστα-παραμέτρων)
```

Παράδειγμα

```
ΚΑΛΕΣΕ Πράξεις (Α, Β, Διαφορά)
```

Λειτουργία

Η εκτέλεση του προγράμματος διακόπτεται και εκτελούνται οι εντολές της διαδικασίας που καλείται. Μετά το τέλος της διαδικασίας η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που ακολουθεί . Η λίστα των παραμέτρων ορίζει τις τιμές που περνούν στη διαδικασία και τις τιμές που αυτή επιστρέφει. Η λίστα παραμέτρων δεν είναι υποχρεωτική.

Στο προηγούμενο παράδειγμα η κλήση των δύο διαδικασιών έγινε με τις εντολές

```
ΚΑΛΕΣΕ Είσοδος_δεδομένων(R)
ΚΑΛΕΣΕ Εκτύπωση(Ε)
```

Σε κάθε περίπτωση κλήσης διαδικασίας μπορεί να γίνεται πέρασμα τιμών μέσω της λίστας παραμέτρων. Πιο συγκεκριμένα, στην περίπτωση της διαδικασίας Είσοδος_δεδομένων γίνεται επιστροφή στο κύριο πρόγραμμα της τιμής της ακτίνας, ενώ στη διαδικασία Εκτύπωση γίνεται μεταβίβαση της τιμής του εμβαδού από το κύριο πρόγραμμα στη διαδικασία. Δηλαδή, η Είσοδος_δεδομένων δέχεται μια τιμή από πληκτρολόγιο, την εκχωρεί στη μεταβλητή Αριθμός και κατά την επιστροφή (μετά το τέλος της διαδικασίας) γίνεται μεταβίβαση αυτής της τιμής στη μεταβλητή R του κύριου προγράμματος. Αντίθετα στη διαδικασία Εκτύπωση κατά την κλήση της μεταβιβάζεται η τιμή της μεταβλητής E του κύριου προγράμματος στη μεταβλητή Αποτέλεσμα της διαδικασίας.

Στο συγκεκριμένο παράδειγμα κάθε διαδικασία έχει από μία παράμετρο. Στη γενική περίπτωση μπορούν να υπάρχουν καμία, μία ή περισσότερες παράμετροι. Οταν υπάρχουν πολλές παράμετροι, τότε άλλες χρησιμοποιούνται για να μεταβιβάσουν τιμές στη διαδικασία και άλλες για να επιστρέψουν τιμές στο κύριο πρόγραμμα.

Κάθε διαδικασία ή συνάρτηση μπορεί να καλείται από το κύριο πρόγραμμα ή από άλλη διαδικασία ή συνάρτηση. Σε κάθε περίπτωση μετά το τέλος της εκτέλεσης της διαδικασίας ή της συνάρτησης γίνεται επιστροφή ακριβώς μετά το σημείο απ' όπου κλήθηκε.

Στη συνέχεια παρουσιάζεται το παράδειγμα 2 υλοποιημένο στις γλώσσες Pascal και Basic.

Προγραμματιστικό περιβάλλον Pascal

```
PROGRAM example2;
   VAR
   r,e:REAL;

FUNCTION  area(r:REAL):REAL;
BEGIN
       area:=pi*sqr(r)
END;

PROCEDURE input(var x:REAL);
BEGIN
   REPEAT
       write ('Δώσε την ακτίνα:');
       readln(x)
   UNTIL x>0;
END;
```

```
PROCEDURE output(result:REAL);
BEGIN
   writeln (Το εμβαδό είναι :',result:6:2)
END;

BEGIN
   input(r);
   e:=area(r);
   output(e)
END.
```

Προγραμματιστικό περιβάλλον Basic

```
' Παράδειγμα 3
DECLARE SUB Eisodos (nb!)
DECLARE SUB Ektypwsh (res!)
DECLARE FUNCTION Emvado! (r!)
CLS
CALL Eisodos(r)
e = Emvado(r)
CALL Ektypwsh(e)
END

SUB Eisodos (nb)
DO
  INPUT "Δώσε την ακτίνα : ", nb
LOOP UNTIL nb > 0
END SUB

SUB Ektypwsh (res)
PRINT "Το εμβαδό του κύκλου είναι :"; res
END SUB

FUNCTION Emvado (r)
pi = 3.14
Emvado = pi * r ^ 2
END FUNCTION
```

10.5.3 Πραγματικές και τυπικές παράμετροι

Η κατανόηση του τρόπου που γίνεται η ανταλλαγή των τιμών ανάμεσα στις παραμέτρους είναι ιδιαίτερα σημαντική και γι αυτό ας παρακολουθήσουμε το επόμενο παράδειγμα.

Παράδειγμα 3

Να γραφεί μια διαδικασία η οποία δέχεται στην είσοδο δύο τιμές και υπολογίζει και επιστρέφει το άθροισμα και τη διαφορά τους.

```
ΠΡΟΓΡΑΜΜΑ Παράδειγμα_3
...
Α<-5
Β<-7
ΚΑΛΕΣΕ Πράξεις(Α, Β, Διαφ1, Αθρ1)
...
Α<-9
Β<-6
ΚΑΛΕΣΕ Πράξεις(Α, Β, Διαφ2, Αθρ2)
...
```

```
ΔΙΑΔΙΚΑΣΙΑ Πράξεις(Χ, Υ, Διαφορά, Άθροισμα)

ΜΕΤΑΒΛΗΤΕΣ
   ΠΡΑΓΜΑΤΙΚΕΣ : Χ, Υ, Διαφορά, Άθροισμα

ΑΡΧΗ
   Διαφορά  <-  Χ-Υ
   Άθροισμα <-  Χ+Υ

ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Οι μεταβλητές Α, Β, Διαφ1, Αθρ1, Διαφ2, Αθρ2 είναι μεταβλητές τού προγράμματος Παράδειγμα_3 και αποτελούν τις **πραγματικές** παραμέτρους, ενώ οι μεταβλητές Χ,Υ, Διαφορά, Άθροισμα είναι μεταβλητές της διαδικασίας Πράξεις, και ονομάζονται **τυπικές** παράμετροι.

Η λίστα των τυπικών παραμέτρων (formal parameter list) καθορίζει τις παραμέτρους στη δήλωση του υποπρογράμματος.

Η λίστα των πραγματικών παραμέτρων (actual parameter list) καθορίζει τις παραμέτρους στην κλήση του υποπρογράμματος.

Οι μεταβλητές Α, Β, Διαφ1 καθώς και όλες οι μεταβλητές του προγράμματος Παράδειγμα_3 δεν είναι γνωστές στη διαδικασία Πράξεις και αντίστοιχα όλες οι μεταβλητές της διαδικασίας Πράξεις είναι άγνωστες στο πρόγραμμα Παράδειγμα_3. Τα ονόματα των τυπικών και των πραγματικών παραμέτρων μπορούν να είναι οποιαδήποτε. Αφού είναι ονόματα μεταβλητών σε διαφορετικά τμήματα προγράμματος, είναι υποχρεωτικά διαφορετικές μεταβλητές, άσχετα αν έχουν το ίδιο όνομα.

Όλες οι μεταβλητές είναι γνωστές, έχουν ισχύ όπως λέγεται, μόνο για το τμήμα προγράμματος στο οποίο έχουν δηλωθεί, ισχύουν δηλαδή **τοπικά** για το συγκεκριμένο υποπρόγραμμα ή κυρίως πρόγραμμα.

Ας παρακολουθήσουμε πώς γίνεται η επικοινωνία ανάμεσα στο πρόγραμμα Παράδειγμα_3 και τη διαδικασία Πράξεις.

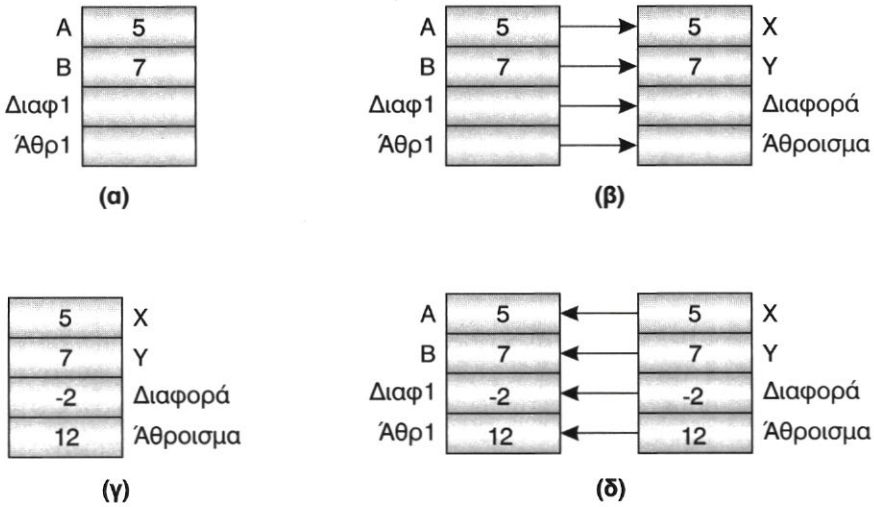
Οι τιμές που υπάρχουν στις μεταβλητές του προγράμματος Α,Β, Διαφ1 και Αθρ1 δίνονται κατά την κλήση στις μεταβλητές της διαδικασίας Χ, Υ, Διαφορά, Άθροισμα.

Έτσι η μεταβλητή Χ παίρνει την τιμή 5 κι η Υ την τιμή 7. Οι μεταβλητές Διαφορά και Άθροισμα δεν παίρνουν καμία τιμή, αφού οι αντίστοιχες μεταβλητές Διαφ1 και Αθρ1 δεν έχουν συγκεκριμένη τιμή.

*Μερικές γλώσσες προγραμματισμού ονομάζουν **ορίσματα** τις τυπικές παραμέτρους και απλά **παραμέτρους** τις πραγματικές παραμέτρους.*

Μετά την εκτέλεση των εντολών της διαδικασίας, όταν εκτελεστεί η εντολή ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ, οι μεταβλητές της διαδικασίας που αναφέρονται στη δήλωση της διαδικασίας δίνουν τις τιμές που περιέχουν στις αντίστοιχες μεταβλητές που περιλαμβάνονται στην κλήση της διαδικασίας Πράξεις. Έτσι η Α παίρνει την τιμή της Χ (=5), η Β την τιμή της Υ (= 7), η Διαφ1 της Διαφορά (=-2) και η μεταβλητή Αθρ1 της Άθροισμα (=12). Με την επιστροφή στο κύριο πρόγραμμα όλες οι θέσεις μνήμης που είχαν δοθεί στη διαδικασία απελευθερώνονται.

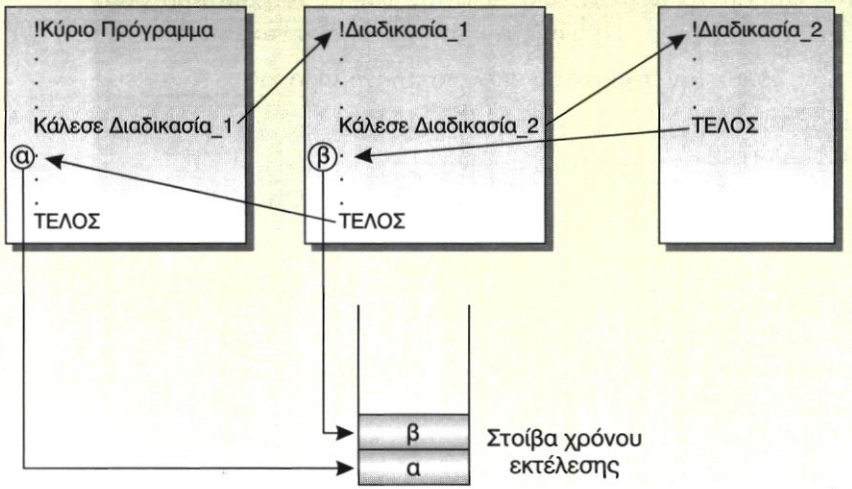
Οι λίστες των παραμέτρων πρέπει να ακολουθούν τους εξής κανόνες:



***Σχ. 10.2.** Πέρασμα παραμέτρων κατά την κλήση διαδικασιών (α) Κατάσταση πριν την κλήση (β) Μεταβίβαση τιμών των μεταβλητών Α και Β στις Χ και Υ αντίστοιχα (γ) Στη διαδικασία εκχωρούνται τιμές στις μεταβλητές Διαφορά και Άθροισμα (δ) Οι τιμές των τελευταίων επιστρέφονται στις Διαφ1 και Αθρ1 μετά το τέλος της διαδικασίας.*

Η χρήση στοίβας στην κλήση διαδικασιών

Η έννοια της στοίβας είναι πολύ χρήσιμη στο ίδιο το λογισμικό των γλωσσών προγραμματισμού. Όταν μία διαδικασία ή συνάρτηση καλείται από το κύριο πρόγραμμα, τότε η αμέσως επόμενη διεύθυνση του κύριου προγράμματος, που ονομάζεται *διεύθυνση επιστροφής* (return address), αποθηκεύεται από το μεταφραστή σε μία στοίβα που ονομάζεται *στοίβα χρόνου εκτέλεσης* (execution time stack). Μετά την εκτέλεση της διαδικασίας ή της συνάρτησης η διεύθυνση επιστροφής απωθείται από τη στοίβα και έτσι ο έλεγχος του προγράμματος μεταφέρεται και πάλι στο κύριο πρόγραμμα. Η τεχνική αυτή εφαρμόζεται και γενικότερα, δηλαδή οποτεδήποτε μία διαδικασία ή συνάρτηση καλεί μία διαδικασία ή συνάρτηση. Για παράδειγμα, έστω ότι μία διαδικασία a καλεί τη διαδικασία b, που με τη σειρά της καλεί τη διαδικασία c κοκ. Στην περίπτωση αυτή οι διευθύνσεις επιστροφής εμφανίζονται στη στοίβα με σειρά c, b, a. Μετά την εκτέλεση κάθε διαδικασίας, η διεύθυνση επιστροφής απωθείται από τη στοίβα και ο έλεγχος μεταβιβάζεται στη διεύθυνση αυτή. Το παράδειγμα αυτό δείχνει μία από τις πολλές χρησιμότητες της LIFO ιδιότητας της στοίβας.



Σχ. 10.3. *Χρήση στοίβας από το μεταφραστή για το χειρισμό κλήσεων διαδικασιών και επιστροφών από αυτές.*

- Ο αριθμός των πραγματικών και των τυπικών παραμέτρων πρέπει να είναι ίδιος.
- Κάθε πραγματική παράμετρος αντιστοιχεί στην τυπική παράμετρο που βρίσκεται στην αντίστοιχη θέση. Για παράδειγμα η πρώτη της λίστας των τυπικών παραμέτρων στην πρώτη της λίστας των πραγματικών παραμέτρων κοκ.
- Η τυπική παράμετρος και η αντίστοιχη της πραγματική πρέπει να είναι του ιδίου τύπου.

10.6. Εμβέλεια μεταβλητών-σταθερών.

Κάθε κύριο πρόγραμμα όπως και κάθε υποπρόγραμμα περιλαμβάνει τις δικές του μεταβλητές και σταθερές.

Οι μεταβλητές αυτές στη **ΓΛΩΣΣΑ** είναι γνωστές στο αντίστοιχο υποπρόγραμμα που δηλώνονται και μόνο σε αυτό. Όλες οι μεταβλητές (και οι σταθερές) είναι **τοπικές** στο συγκεκριμένο τμήμα προγράμματος.

Ο μόνος τρόπος για να περάσει μία τιμή από ένα υποπρόγραμμα σε ένα άλλο ή από το κυρίως πρόγραμμα σε ένα υποπρόγραμμα είναι δια μέσου των παραμέτρων κατά το στάδιο της κλήσης του υποπρογράμματος και μετά το τέλος της εκτέλεσης του υποπρογράμματος.

Ας δούμε τον παρακάτω σκελετό προγράμματος

```
ΠΡΟΓΡΑΜΜΑ Αρχικό
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Α, Β, Γ
ΑΡΧΗ
...
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
ΔΙΑΔΙΚΑΣΙΑ Πρώτη
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ : Δ, Ε, Ζ, Η
ΑΡΧΗ
...
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
ΔΙΑΔΙΚΑΣΙΑ Δεύτερη
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ : Γ, Θ, Ι
ΑΡΧΗ
...
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑΣ
```

Οι μεταβλητές του προγράμματος Αρχικό με ονόματα Α, Β, Γ είναι γνωστές, ισχύουν μόνο για το πρόγραμμα. Έξω από το πρόγραμμα σε όλα τα υποπρογράμματα οι μεταβλητές αυτές δεν ισχύουν. Επίσης η διαδικασία Πρώτη έχει τις πραγματικές μεταβλητές Δ, Ε, Ζ, Η, οι οποίες ισχύουν μόνο για τη συγκεκριμένη διαδικασία και όχι για τα υπόλοιπα υποπρογράμματα ή το κύριο πρόγραμμα.

Η διαδικασία Δεύτερη έχει και αυτή τις δικές της μεταβλητές Γ,Θ,Ι. Οι μεταβλητές αυτές ισχύουν μόνο για τη διαδικασία Δεύτερη. Η μεταβλητή με το όνομα Γ δεν έχει καμία σχέση με τη μεταβλητή Γ του κύριου προγράμματος, η μία είναι τύπου Ακεραίου και η άλλη τύπου Πραγματικού. Αφού όλες οι μεταβλητές είναι τοπικές, το ίδιο όνομα μεταβλητής μπορεί να εμφανίζεται σε διαφορετικά τμήματα προγράμματος, χωρίς να αντιστοιχεί στην ίδια μεταβλητή. Το ίδιο έγινε και με τη μεταβλητή R της ακτίνας του κύκλου στο παράδειγμα 2.

Ότι ισχύει για τις μεταβλητές ισχύει και για τις σταθερές.

*Το τμήμα του προγράμματος που ισχύουν οι μεταβλητές λέγεται **εμβέλεια** (scope) μεταβλητών.*

Πολλές γλώσσες προγραμματισμού επιτρέπουν τη χρήση των μεταβλητών και των σταθερών, όχι μόνο στο τμήμα προγράμματος που δηλώνονται, αλλά και σε άλλα ή ακόμη και σε όλα τα υπόλοιπα υποπρογράμματα. Αυτό που καθορίζει την περιοχή που ισχύουν οι μεταβλητές και οι σταθερές είναι η εμβέλεια των μεταβλητών της γλώσσας

Απεριόριστη εμβέλεια.

Σύμφωνα με αυτή την αρχή όλες οι μεταβλητές και όλες οι σταθερές είναι γνωστές και μπορούν να χρησιμοποιούνται σε οποιοδήποτε τμήμα του προγράμματος, άσχετα που δηλώθηκαν. 'Ολες οι μεταβλητές είναι **καθολικές**.

Η απεριόριστη εμβέλεια καταστρατηγεί την αρχή της αυτονομίας των υποπρογραμμάτων, δημιουργεί πολλά προβλήματα και τελικά είναι αδύνατη για μεγάλα προγράμματα με πολλά υποπρογράμματα, αφού ο καθένας που γράφει κάποιο υποπρόγραμμα πρέπει να γνωρίζει τα ονόματα όλων των μεταβλητών που χρησιμοποιούνται στα υπόλοιπα υποπρογράμματα.

Περιορισμένη εμβέλεια

Η περιορισμένη εμβέλεια υποχρεώνει όλες τις μεταβλητές που χρησιμοποιούνται σε ένα τμήμα προγράμματος, να δηλώνονται σε αυτό το τμήμα. Όλες οι μεταβλητές είναι **τοπικές,** ισχύουν δηλαδή για το υποπρόγραμμα στο οποίο δηλώθηκαν. Στη **ΓΛΩΣΣΑ** έχουμε περιορισμένη εμβέλεια.

Τα πλεονεκτήματα της περιορισμένης εμβέλειας είναι η απόλυτη αυτονομία όλων των υποπρογραμμάτων και η δυνατότητα να χρησιμοποιείται οποιοδήποτε όνομα, χωρίς να ενδιαφέρει αν το ίδιο χρησιμοποιείται σε άλλο υποπρόγραμμα.

Μερικώς περιορισμένη εμβέλεια.

Σύμφωνα με αυτή την αρχή άλλες μεταβλητές είναι τοπικές και άλλες καθολικές. Κάθε γλώσσα προγραμματισμού έχει τους δικούς της κανόνες και μηχανισμούς για τον τρόπο και τις προϋποθέσεις που ορίζονται οι μεταβλητές ως τοπικές ή καθολικές.

Η μερικώς περιορισμένη εμβέλεια προσφέρει μερικά πλεονεκτήματα στον πεπειραμένο προγραμματιστή, αλλά για τον αρχάριο περιπλέκει το πρόγραμμα δυσκολεύοντας την ανάπτυξή του.

10.7. Αναδρομή

Ενα υποπρόγραμμα καλείται από το κύριο πρόγραμμα ή άλλο υποπρόγραμμα. Υπάρχει όμως και η δυνατότητα ένα υποπρόγραμμα να καλεί τον εαυτό του. Η δυνατότητα αυτή αποκαλείται αναδρομή.

Η έννοια της αναδρομής παρουσιάστηκε στο κεφάλαιο 3, όπου δόθηκαν παραδείγματα και έγινε μία πρώτη αναφορά στα πλεονεκτήματα, αλλά και τα προβλήματα που παρουσιάζει η χρήση αναδρομικών αλγορίθμων.

Αναδρομή ονομάζεται η δυνατότητα ενός υποπρογράμματος να καλεί τον εαυτό του.

Η αναδρομή υλοποιείται στον προγραμματισμό με χρήση αναδρομικών υποπρογραμμάτων και επειδή τόσο η αναδρομή ως έννοια όσο και η λειτουργία της είναι αρκετά πολύπλοκες, θα παρουσιαστούν αναλυτικά στη συνέχεια.

Ο ορισμός κάθε αναδρομικού υποπρογράμματος έχει δύο τμήματα, την αναδρομική σχέση και την τιμή βάσης ή συνθήκη τερματισμού

Ας εξετάσουμε το παράδειγμα υπολογισμού του παραγοντικού:

Το παραγοντικό ορίζεται αναδρομικά ως εξής:

$$n! = \begin{cases} n \cdot (n-1)! & \text{αν } n > 0 \\ 1 & \text{αν } n = 0 \end{cases}$$

Τιμή βάσης ή συνθήκη τερματισμού είναι η τιμή που ορίζεται για μία συγκεκριμένη τιμή της παραμέτρου της συνάρτησης, στο παράδειγμα η τιμή είναι 1 για n=0.

Αναδρομική σχέση είναι η n*(n-1)!, όπου η τιμή της συνάρτησης υπολογίζεται με βάση τις προηγούμενες τιμές της συνάρτησης, οι οποίες πρέπει να υπολογιστούν.

Οι συναρτήσεις στη γλώσσα που υλοποιούν τον υπολογισμό του παραγοντικού τόσο αναδρομικά όσο και επαναληπτικά είναι οι εξής.

```
ΣΥΝΑΡΤΗΣΗ Παραγοντικο(N):ΑΚΕΡΑΙΑ
!Υπολογισμός του παραγοντικού με αναδρομική διαδικασία
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: N
ΑΡΧΗ
    ΑΝ N=0 ΤΟΤΕ
        Παραγοντικο <- 1
    ΑΛΛΙΩΣ
        Παραγοντικο <- N*Παραγοντικο(N-1)
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ

ΣΥΝΑΡΤΗΣΗ Παραγοντικο(N):ΑΚΕΡΑΙΑ
!Υπολογισμός του παραγοντικού με επαναληπτική διαδικασία
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, N
ΑΡΧΗ
    Fact <- 1
    ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ N
        Fact <- Fact*i
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΣΥΝΑΡΤΗΣΗΣ
```

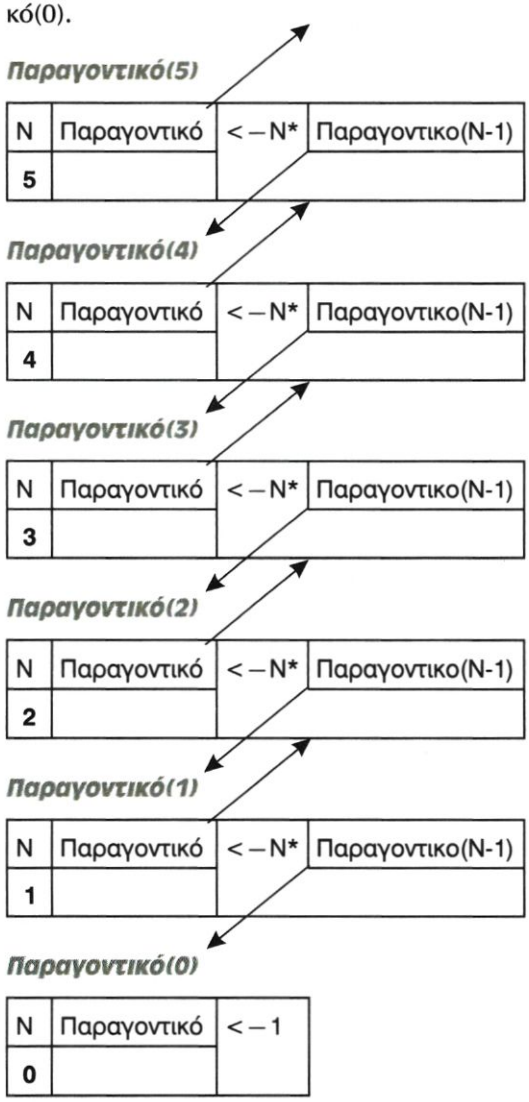
Και οι δύο αυτές συναρτήσεις, η πρώτη αναδρομική και η δεύτερη επαναληπτική έχουν το ίδιο αποτέλεσμα. Το αναδρομικό υποπρόγραμμα επειδή εκφράζει ακριβώς την αλγεβρική συνάρτηση υπολογισμού του παραγοντικού είναι πιο απλό στη διατύπωση, αλλά είναι πιο δύσκολο στην κατανόηση του τρόπου που λειτουργεί.

Γενικά η έννοια της αναδρομής καθώς και ο τρόπος με τον οποίο εκτελείται μία αναδρομική διαδικασία από τον υπολογιστή παρουσιάζει στην αρχή δυσκολία στην κατανόηση της.

Ας παρακολουθήσουμε τον τρόπο με τον οποίο υπολογίζεται το 5! με τη χρήση της αναδρομικής συνάρτησης Παραγοντικο().

Εφόσον η παράμετρος της συνάρτησης είναι αρχικά 5, διάφορη δηλαδή του 0, καλείται η συνάρτηση Παραγοντικό(4). Στην συνέχεια επειδή η πα-

ράμετρος είναι διάφορη του μηδενός, καλείται το Παραγοντικό(3) και η διαδικασία συνεχίζεται όπως δείχνει το σχήμα μέχρι την κλήση του Παραγοντικό(0).



Για τη τιμή της παραμέτρου N=0 το Παραγοντικό(0) έχει τιμή 1, την τιμή βάσης. Έτσι αρχίζει η αντίστροφη διαδικασία υπολογισμού διαδοχικά του υπολογισμού των συναρτήσεων Παραγοντικό(2)=1*2, Παραγοντικό(3)=2*3, Παραγοντικό(4)=6*4 όπως δείχνει το παρακάτω σχήμα.

Το τελευταίο βήμα είναι ο υπολογισμός του Παραγοντικό(5)=24*5. Η τιμή αυτή δηλαδή 120 είναι η τιμή που η συνάρτηση τελικά επιστρέφει.

Παραγοντικό(5)

N	Παραγοντικό	<—N*	Παραγοντικο(N-1)
5	120		

Παραγοντικό(4)

N	Παραγοντικό	<—N*	Παραγοντικο(N-1)
4	24		

Παραγοντικό(3)

N	Παραγοντικό	<—N*	Παραγοντικο(N-1)
3	6		

Παραγοντικό(2)

N	Παραγοντικό	<—N*	Παραγοντικο(N-1)
2	2		

Παραγοντικό(1)

N	Παραγοντικό	<—N*	Παραγοντικο(N-1)
1	1		

Παραγοντικό(0)

N	Παραγοντικό	<—1
0	1	

Ο μηχανισμός διαχείρισης των αναδρομικών διαδικασιών από το μεταφραστή κάνει χρήση μιας στοίβας. Κατά την κλήση μιας αναδρομικής διαδικασίας από την ίδια φυλάσσονται σε μία στοίβα όλες οι τιμές των μεταβλητών, που έχουν πριν την κλήση (λειτουργία ώθηση). Κατά την επιστροφή ανασύρονται οι τιμές αυτές από τη στοίβα (λειτουργία απώθηση) και έτσι η διαδικασία μπορεί να συνεχίσει τη λειτουργία της με τις προηγούμενες τιμές. Προφανώς, αν υπάρχουν διαδοχικές κλήσεις της διαδικασίας, λόγω της ιδιότητας LIFO με την οποία λειτουργεί η στοίβα, εξασφαλίζεται ότι εξάγονται από αυτή οι πλέον πρόσφατες τιμές των μεταβλητών, δηλαδή αυτές που αποθηκεύτηκαν κατά την τελευταία κλήση.

Πολύ συχνά προβλήματα τα οποία μπορούν να λυθούν με χρήση αναδρομής λύνονται και με απλούς επαναληπτικούς αλγόριθμους. Παραδείγματα παρουσιάστηκαν στο κεφάλαιο 3, όπου έγινε και σύγκριση μεταξύ των δύο μεθόδων. Εδώ επαναλαμβάνουμε ότι αν και η αναδρομή φαίνεται ως πιο φυσικός τρόπος προγραμματισμού, πρέπει να χρησιμοποιείται με μέτρο, γιατί το κέρδος σε χρόνο προγραμματισμού δημιουργεί συχνά απώλεια σε χρόνο εκτέλεσης.

Προγραμματιστικό περιβάλλον Pascal

```
FUNCTION factorial(n:INTEGER) : INTEGER;
BEGIN
    IF n=0 THEN
        factorial:=1
    ELSE
        factorial:=n*factorial(n-1);
END;

FUNCTION factorial(n:INTEGER) : INTEGER;
VAR
    i,fact:integer;
BEGIN
    fact:=1;
    FOR i:=2 TO n DO
        fact:=i*fact;
        factorial:=fact
END;
```

Προγραμματιστικό περιβάλλον Basic

```
FUNCTION Factorial (n)
IF n = 0 THEN
   Factorial = 1
ELSE
   Factorial = n * Factorial(n - 1)
END IF
END FUNCTION


FUNCTION Factorial (n)
fact = 1
FOR i = 2 TO n
  fact = fact * i
NEXT i
Factorial = fact
END FUNCTION
```

Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάστηκαν οι αρχές και τα χαρακτηριστικά του τμηματικού προγραμματισμού και ο τρόπος που χειρίζεται τα υποπρογράμματα η ΓΛΩΣΣΑ.

Η χρήση υποπρογραμμάτων σε ένα πρόγραμμα παρουσιάζει πολλά πλεονεκτήματα, αφού διευκολύνει την ανάπτυξη των αλγόριθμων και την υλοποίηση του προγράμματος σε λιγότερο μάλιστα χρόνο και διευκολύνει την κατανόηση και τη διόρθωσή του. Υπάρχουν δύο ειδών υποπρογράμματα, οι συναρτήσεις και οι διαδικασίες. Η συνάρτηση παράγει ένα αποτέλεσμα και επιστρέφει μόνο μία τιμή, ενώ οι διαδικασίες μπορούν να εκτελούν όλες τις ενέργειες ενός προγράμματος. Τα υποπρογράμματα επικοινωνούν μεταξύ τους καθώς και με το κύριο πρόγραμμα με τη βοήθεια των παραμέτρων. Στη δήλωση του υποπρογράμματος βρίσκονται οι τυπικές παράμετροι, ενώ οι πραγματικές παράμετροι βρίσκονται στην εντολή κλήσης του υποπρογράμματος. Οι διαδικασίες ενεργοποιούνται στη ΓΛΩΣΣΑ με την εντολή ΚΑΛΕΣΕ, ενώ οι συναρτήσεις μόνο με την αναγραφή του ονόματός τους μέσα σε μία έκφραση.

Η δυνατότητα ενός υποπρογράμματος να καλεί τον εαυτό του, ονομάζεται αναδρομή. Πολύ συχνά τα προβλήματα που λύνονται με αναδρομή μπορούν να λυθούν με χρήση απλής επανάληψης.

Λέξεις κλειδιά

Τμήμα προγράμματος, διαδικασία, συνάρτηση, παράμετρος, εμβέλεια παραμέτρων, αναδρομή.

Ερωτήσεις - Θέματα για συζήτηση

1. Πως ορίζεται ο τμηματικός προγραμματισμός;
2. Ποια τα βασικά χαρακτηριστικά των υποπρογραμμάτων;
3. Ποια η διαφορά παραμέτρων και απλών μεταβλητών;
4. Ποια η βασική διαφορά διαδικασιών και συναρτήσεων;
5. Πώς εκτελείται μία συνάρτηση;
6. Πώς καλείται μία διαδικασία;
7. Ποια η διαφορά τυπικών και πραγματικών παραμέτρων;

8. Πώς γίνεται η ανταλλαγή των τιμών ανάμεσα στις τυπικές και πραγματικές παραμέτρους; Δώσε ένα παράδειγμα ανταλλαγής παραμέτρων.
9. Τι ονομάζεται εμβέλεια μεταβλητών;
10. Τι είναι η αναδρομή;
11. Ποια είναι τα τμήματα που περιλαμβάνει κάθε αναδρομικός ορισμός; Δώσε ένα παράδειγμα.

Βιβλιογραφία

1. Θ. Αλεβίζος, Α. Καμπουρέλης, *Εισαγωγή με τη γλώσσα Pascal*, Αθήνα, 1984.
2. Γ. Βουτυράς, *Basic: Αλγόριθμοι και εφαρμογές*, Κλεψύδρα, Αθήνα, 1991
3. Χρ. Κοίλιας, *Η QuickBasic και οι εφαρμογές της*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.
4. R. Shackelford, *Introduction to Computing and Algorithms*, Addison-Wesley, USA, 1998.
5. S. Leestma-L.Nyhoff, *Turbo Pascal, Programming and Solving*, McMillan, New York, 1990.
6. N. Wirth, *Systematic Programming: An introduction*, Prentice Hall, 1973.

ΚΕΦΑΛΑΙΟ 11ο

ΣΥΓΧΡΟΝΑ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΑ ΠΕΡΙΒΑΛΛΟΝΤΑ

ΠΕΡΙΕΧΟΜΕΝΑ

- Αντικειμενοστραφής προγραμματισμός
- Οδηγούμενος από γεγονότα προγραμματισμός
- Υλοποίηση εφαρμογών σε σύγχρονο προγραμματιστικό περιβάλλον
- Στοιχεία γραφικού προγραμματιστικού περιβάλλοντος
- Επικοινωνία με άλλες εφαρμογές

Εισαγωγή

Η ανάγκη δημιουργίας σύγχρονων περιβαλλόντων εργασίας, προσανατολισμένων στη φιλικότητα επικοινωνίας με τον χρήστη, έφερε δραστικές αλλαγές στον προγραμματισμό και τα προγραμματιστικά εργαλεία. Σε ένα σύγχρονο προγραμματιστικό περιβάλλον οι έννοιες του αντικειμενοστραφούς, του οδηγούμενου από γεγονότα προγραμματισμού και της οπτικής σχεδίασης είναι βασικά χαρακτηριστικά. Παρέχονται μια σειρά από γραφικά εργαλεία για την πλήρη υποστήριξη και ευκολότερη και ταχύτερη ανάπτυξη προγραμμάτων. Ιδιαίτερο γνώρισμα των περιβαλλόντων αυτών είναι η έμφαση που δίνεται στο γραφικό περιβάλλον και στην επικοινωνία της εφαρμογής με το χρήστη. Ακόμη οι εφαρμογές δεν είναι απομονωμένες, αλλά συνεργάζονται με άλλες εφαρμογές και ετερογενή περιβάλλοντα.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να ορίζουν τις έννοιες αντικείμενο, ιδιότητα, γεγονός, μέθοδος,
- να διατυπώνουν τη μεθοδολογία σχεδιασμού μιας εφαρμογής σε ένα σύγχρονο προγραμματιστικό περιβάλλον,
- να αναλύουν μια εφαρμογή μέσα από αντικείμενα και γεγονότα,
- να κατασκευάζουν απλά προγράμματα σε σύγχρονα προγραμματιστικά περιβάλλοντα συνδυάζοντάς τα με τις γνώσεις τους στον τμηματικό και στον δομημένο προγραμματισμό.

Προερωτήσεις

- ✓ ποιες είναι οι τεχνικές προγραμματισμού που εφαρμόζονται σε ένα σύγχρονο προγραμματιστικό περιβάλλον;
- ✓ είναι εφικτό να χρησιμοποιήσουμε διαφορετικές τεχνικές προγραμματισμού στην ανάπτυξη μιας εφαρμογής;
- ✓ η εφαρμογή νέων τεχνικών προγραμματισμού καταργεί τις υπάρχουσες;
- ✓ όταν αναφερόμαστε σε εφαρμογή νέων τεχνικών προγραμματισμού κατ' ανάγκη δυσκολεύεται το έργο του προγραμματιστή;
- ✓ ποια τα πλεονεκτήματα ενός σύγχρονου προγραμματιστικού περιβάλλοντος;
- ✓ είναι εφικτό να επιτύχουμε τη συνεργασία διαφορετικών εφαρμογών σε ένα σύγχρονο περιβάλλον εργασίας;

11.1. Αντικειμενοστραφής προγραμματισμός

Τα κλασικά προγραμματιστικά περιβάλλοντα εργασίας, οδήγησαν στην ανάπτυξη πολύπλοκων προϊόντων λογισμικού, που η κατασκευή τους και η συντήρησή τους απαιτούσε ειδικούς με βαθιές γνώσεις σε επιμέρους θέματα, όπως τεχνικές διαχείρισης μνήμης, πρότυπα επικοινωνίας, ειδικές γλώσσες προγραμματισμού και λειτουργικά συστήματα. Η πολυπλοκότητα κατασκευής του λογισμικού δημιούργησε ταυτόχρονα προβληματισμούς και δρομολόγησε απόπειρες εξεύρεσης φιλικότερων τεχνικών και μεθόδων σχεδιασμού προγραμμάτων. Από την άλλη πλευρά η μοντελοποίηση του ανθρώπινου συλλογισμού μέσω της τεχνητής νοημοσύνης, έκανε φανερή την ανάγκη της δημιουργίας ενός διαφορετικού προγραμματιστικού περιβάλλοντος που θα ομαδοποιούσε στην ίδια οντότητα όλες τις πληροφορίες και ιδιότητες που αφορούσαν στην ίδια έννοια.

Ο αντικειμενοστραφής σχεδιασμός προγραμμάτων γεννιέται από την προσπάθεια αντιμετώπισης αυτών ακριβώς των ζητημάτων. Χρησιμοποιώντας τον όρο **αντικειμενοστραφής προγραμματισμός** (object - oriented programming) δεν αναφερόμαστε σε κάποιο συγκεκριμένο προϊόν ή μια γλώσσα προγραμματισμού, αλλά σε ένα διαφορετικό τρόπο προσέγγισης προβλημάτων με τον υπολογιστή.

Ο αντικειμενοστραφής προγραμματισμός ή η αντικειμενοστραφής σχεδίαση προέκυψε από τη στιγμή που απαντήθηκε διαφορετικά – από ότι απαντιόταν μέχρι τότε - το ερώτημα *"Η δομή των προγραμμάτων είναι προτιμότερο να στηρίζεται στις "ενέργειες" ή στα δεδομένα;"*. Η νέα απάντηση που δόθηκε, η απάντηση *"δεδομένα"*, προσδιορίζει και τη βασική διαφορά ανάμεσα στις παραδοσιακές προγραμματιστικές τεχνικές και στην αντικειμενοστραφή προσέγγιση η οποία περιγράφει *"ενέργειες"* (επεξεργασία) που εφαρμόζονται πάνω σε δεδομένα.

Η αντικειμενοστραφής σχεδίαση λοιπόν εκλαμβάνει σαν πρωτεύοντα δομικά στοιχεία ενός προγράμματος τα δεδομένα, από τα οποία δημιουργούνται με κατάλληλη μορφοποίηση τα αντικείμενα. Αυτή η σχεδίαση αποδείχθηκε ότι επιφέρει καλύτερα αποτελέσματα, αφού τα προγράμματα που δημιουργούνται είναι περισσότερο ευέλικτα, επαναχρησιμοποιήσιμα και περισσότερο φιλικά.

Οι θεμελιώδεις αρχές του αντικειμενοστραφούς προγραμματισμού, πηγάζουν από τον καθημερινό μας φυσικό κόσμο. Στηρίζονται στο γεγονός, ότι για να μπορέσει κάποιος να κατανοήσει άγνωστες σε αυτόν έννοιες, θα πρέπει να καθοδηγηθεί μέσω της προσομοίωσης των άγνωστων αυτών εννοιών αντιστοιχίζοντας αυτές σε πρακτικές γνώσεις και εικόνες από το πε-

Μεταξύ των πρώτων αντικειμενοστραφών γλωσσών προγραμματισμού που εμφανίστηκαν είναι η Simula και η Smalltalk.

ριβάλλον του, τις οποίες γνωρίζει και μπορεί πολύ εύκολα να χειριστεί. Σύμφωνα με την αντικειμενοστραφή θεωρεία ανάπτυξης εφαρμογών, η προσέγγιση κάθε προβλήματος πρέπει να γίνεται με φυσική ερμηνεία και να μην στηρίζεται σε πολύπλοκα τεχνικά ζητήματα.

Σύγχρονα αντικειμενοστραφή προγραμματιστικά περιβάλλοντα είναι η Visual C++, η Java, η Visual Basic, το Delphi.

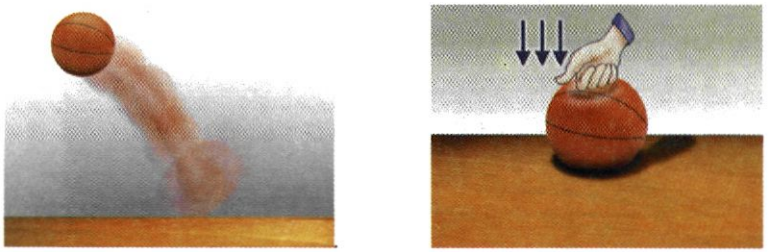
11.1.1. Αντικείμενα

Βασικά στοιχεία του αντικειμενοστραφούς προγραμματισμού αποτελούν τα **αντικείμενα** (objects). Αν ρίξουμε μια ματιά γύρω μας, θα παρατηρήσουμε ότι το περιβάλλον μας αποτελείται από αντικείμενα, τα οποία μπορούμε πολύ εύκολα να αντιληφθούμε και να χειριστούμε. Ως παραδείγματα αντικειμένων μπορούμε να αναφέρουμε ένα αυτοκίνητο, μια μπάλα καλαθοσφαίρισης, ένα παγκάκι, ένα πορτοκάλι και έναν άνθρωπο (σχήμα 11.1).

***Σχ. 11.1.** Αντικείμενα από το φυσικό περιβάλλον*

Κάθε αντικείμενο είναι ευδιάκριτο και αυτάρκες, ενώ το σύνολο των χαρακτηριστικών ιδιοτήτων που προσδιορίζουν με λεπτομέρεια τη φυσική του υπόσταση. Για παράδειγμα, μια μπάλα καλαθοσφαίρισης έχει σχήμα, είναι κατασκευασμένη από κάποιο υλικό, έχει κάποιο χρώμα. Ο άνθρωπος έχει επίσης χαρακτηριστικά (ιδιότητες) όπως όνομα, βάρος, ημερομηνία, γέννησης, χρώμα ματιών κ.λπ. Είναι εμφανές ότι η περιγραφή ενός αντικειμένου καθορίζεται από τις τιμές των επιμέρους ιδιοτήτων του. Για παράδειγμα, το χρώμα της μπάλας καλαθοσφαίρισης είναι πορτοκαλί, το σχήμα της είναι στρογγυλό, το χρώμα των ματιών του ανθρώπου είναι καστανό.

Αν παρατηρήσουμε λεπτομερέστερα ένα αντικείμενο, θα διαπιστώσουμε ότι περιέχει και κανόνες συμπεριφοράς, "γνωρίζει" δηλαδή πως πρέπει να αντιδράσει όταν μια ενέργεια ασκηθεί επάνω του. Αν ρίξουμε τη μπάλα καλαθοσφαίρισης από κάποιο ύψος, αυτή θα αναπηδήσει στο πάτωμα, αν την πιέσουμε θα αλλοιωθεί το σχήμα της (σχήμα 11.2).

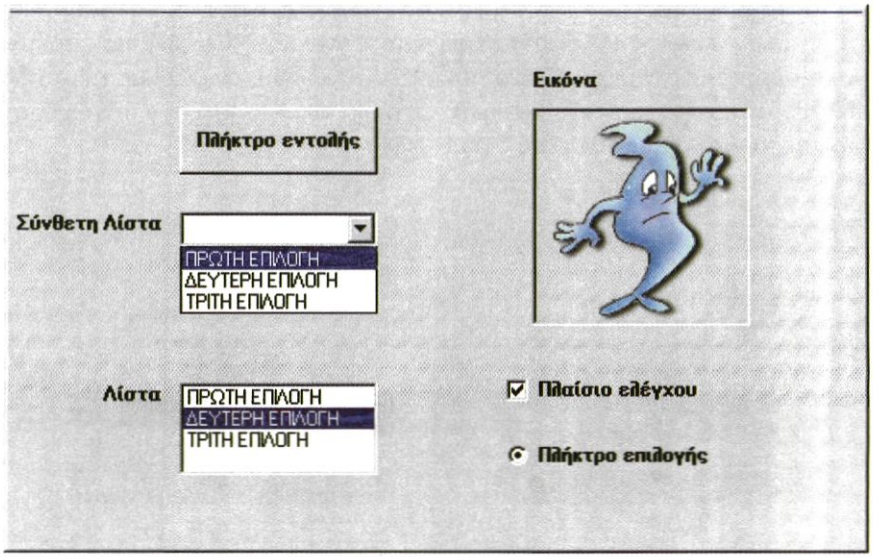


Σχ. 11.2 . Συμπεριφορά αντικειμένου μπάλας καλαθοσφαίρισης

Πως είναι όμως δυνατό να συνδυάσουμε τον καθημερινό μας κόσμο με τον κόσμο της Πληροφορικής; Μα φυσικά με την προσομοίωση των συστατικών μιας εφαρμογής ως φυσικά αντικείμενα.

Σε μια εφαρμογή, ένα αντικείμενο είναι ο ομαδοποιημένος συνδυασμός κώδικα και δεδομένων, τα οποία έχουμε τη δυνατότητα να τα χειριστούμε ενιαία. Από την μεριά ενός προγραμματιστή, τα δεδομένα (data) αποτελούν τα χαρακτηριστικά και οι ενέργειες (operations) καθορίζουν τη συμπεριφορά ενός αντικειμένου. Οι ενέργειες στον αντικειμενοστραφή προγραμματισμό αναφέρονται και ως **μέθοδοι** (methods).

Το "χτίσιμο" μιας αντικειμενοστραφούς εφαρμογής , επιτυγχάνεται με τη δημιουργία και το χειρισμό αντικειμένων. Σε μερικά αντικειμενοστραφή προγραμματιστικά περιβάλλοντα τα αντικείμενα της εφαρμογής μπορούν να δημιουργούνται είτε μέσω κώδικα είτε με τη βοήθεια κατάλληλων γραφικών εργαλείων, σε άλλα πάλι, δημιουργούνται μόνο μέσω κώδικα. Τα αντικείμενα μιας εφαρμογής μπορεί να είναι είτε γραφικά, είτε όχι. Μερικά γραφικά αντικείμενα που υποστηρίζουν πολλά αντικειμενοστραφή προγραμματιστικά περιβάλλοντα, όπως μια φόρμα, ένα πλήκτρο εντολής, ένα πλαίσιο λίστας, ένα πλαίσιο σύνθετης λίστας, ένα πλαίσιο ελέγχου, ένα πλήκτρο εντολής και μια εικόνα φαίνονται στο σχήμα 11.3.



***Σχ. 11.3** . Γραφικά αντικείμενα μιας εφαρμογής*

Πολλά από τα σύγχρονα προγραμματιστικά αντικειμενοστραφή περιβάλλοντα, προσφέρουν ένα αριθμό προκαθορισμένων αντικειμένων, τα οποία μπορούν να αξιοποιηθούν μέσα στην εφαρμογή. Φυσικά δεν περιορίζουν τον προγραμματιστή στο χειρισμό αυτών και μόνο των προκαθορισμένων αντικειμένων, αλλά του επιτρέπουν, μέσω εντολών προγράμματος, το σχεδιασμό νέων αντικειμένων προσαρμοσμένων στις δικές του ανάγκες.

Σε μια εφαρμογή μπορούμε να συμπεριλάβουμε και μη γραφικά αντικείμενα. Αυτό σημαίνει ότι το αντικείμενο υπάρχει, αλλά δεν έχει εξωτερικά χαρακτηριστικά. Τέτοια ειδικά αντικείμενα σε μια εφαρμογή μπορεί να προσφέρονται έτοιμα από το περιβάλλον ανάπτυξης, αλλά μπορεί και ο προγραμματιστής να δημιουργήσει και καινούργια αντικείμενα, μέσα από τμήματα κώδικα.

11.1.2. Κλάσεις

Όπως ήδη αναφέραμε ένα αντικείμενο είναι αυτάρκες, έχει δηλαδή τη δική του φυσική υπόσταση και ταυτότητα. Αυτό πρακτικά σημαίνει, ότι ακόμη και αν δύο αντικείμενα έχουν ακριβώς τις ίδιες τιμές στις ιδιότητές τους εξακολουθούν να παραμένουν δύο ανεξάρτητα αντικείμενα. Για παράδειγμα δύο μπάλες καλαθοσφαίρισης έχουν τα ίδια χαρακτηριστικά, κάθε φορά όμως σε ένα παιχνίδι χρησιμοποιείται μόνο μια από αυτές. Μπο-

ρούμε να χαρακτηρίσουμε την έννοια μπάλα καλαθοσφαίρισης ως το πρότυπο δημιουργίας ανεξάρτητων αντικειμένων μπάλας. Κάθε ανεξάρτητο αντικείμενο, αποτελεί ένα στιγμιότυπο (instance) του γενικού τύπου μπάλα καλαθοσφαίρισης. Ο γενικός τύπος ενός αντικειμένου καλείται **κλάση** (class) και καθορίζει τις αρχικές ιδιότητες και τη συμπεριφορά κάθε αντικειμένου που προέρχεται από αυτή.

Στη Smalltalk, αντικειμενοστραφή γλώσσα προγραμματισμού, καθετί, συμπεριλαμβανομένων των κλάσεων, θεωρείται αντικείμενο. Η κλάση εκλαμβάνεται σαν ένα στιγμιότυπο μιας κλάσης ανωτέρου επιπέδου που καλείται μετακλάση (metaclass).

Σε ένα αντικειμενοστραφές προγραμματιστικό περιβάλλον η υποστήριξη κλάσεων αποτελεί κυρίαρχο στοιχείο. Η κλάση είναι η στατική περιγραφή ενός συνόλου αντικειμένων. Όλα τα αντικείμενα δημιουργούνται ως ακριβή αντίγραφα της κλάσης τους. Για παράδειγμα, διαφορετικά αντικείμενα εικόνας, αποτελούν στιγμιότυπα της κλάσης εικόνα.

11.1.3. Ιδιότητες

Κατά τη δημιουργία τους τα διαφορετικά αυτά αντικείμενα έχουν τις ίδιες ιδιότητες και αντιδρούν με τον ίδιο τρόπο σε όποιο μήνυμα ανιχνευτεί από το περιβάλλον τους. Καθένα από τα αντικείμενα εικόνας κληρονομεί τις αρχικές ιδιότητες και την συμπεριφορά του από την κλάση εικόνα. Ωστόσο ο προγραμματιστής έχει τη δυνατότητα, μετατρέποντας τις τιμές των **ιδιοτήτων** τους (attributes ή properties), να διαμορφώσει διαφορετικά την εξωτερική εμφάνιση των αντικειμένων. Μπορεί επίσης με τη συγγραφή των κατάλληλων εντολών κώδικα να αντιστοιχήσει διαφορετικές εργασίες σε καθένα από αυτά τα αντικείμενα εικόνας.

Όπως ακριβώς είναι δυνατόν στο φυσικό μας κόσμο να μετατρέψουμε τα χαρακτηριστικά ενός αντικειμένου για παράδειγμα να αλλάξουμε το χρώμα ενός ποδηλάτου, ή τη θέση ενός φωτιστικού, έτσι και σε μια αντικειμενοστραφή εφαρμογή έχουμε τη δυνατότητα ή να αλλάξουμε το χρώμα, τη θέση, το μέγεθος ή όποιο άλλο χαρακτηριστικό του αντικείμενου επιθυμούμε.

Σε ένα γραφικό αντικειμενοστραφές περιβάλλον ανάπτυξης εφαρμογών, οι ιδιότητες των αντικειμένων είναι δυνατό να πάρουν αρχικές τιμές κατά το σχεδιασμό, αλλά τις περισσότερες φορές έχουμε τη δυνατότητα να τις μετατρέψουμε και κατά την εκτέλεση της εφαρμογής.

Κατά το σχεδιασμό μιας εφαρμογής πολλά από τα σύγχρονα περιβάλλοντα ανάπτυξης εφαρμογών μας παρέχουν κατάλληλα εργαλεία, όπως **παράθυρο ιδιοτήτων** (properties window), που εμφανίζουν τη τιμή κάθε ιδιότητας ενός αντικειμένου και μας επιτρέπουν τη μετατροπή τους.

Όταν θέλουμε να μετατρέψουμε τη τιμή μιας ιδιότητας κατά την εκτέλεση μιας εφαρμογής, επειδή δεν έχουμε πρόσβαση στα εργαλεία που μας

Χαρακτηριστικά αντικειμενοστραφών περιβαλλόντων ανάπτυξης εφαρμογών

Το βασικότερο χαρακτηριστικό ενός αντικειμενοστραφούς περιβάλλοντος ανάπτυξης εφαρμογών είναι, όπως ήδη προαναφέραμε, η **κλάση**. Ένα πραγματικά αντικειμενοστραφές περιβάλλον ανάπτυξης εφαρμογών πρέπει να διαθέτει ακόμη τα παρακάτω χαρακτηριστικά :

Υποστήριξη αφηρημένων τύπων (Data abstraction). Κάθε αντικείμενο περιγράφεται ως υλοποίηση αφηρημένων τύπων. Κάθε αφηρημένος τύπος έχει ως δομικά του στοιχεία δεδομένα και συγκεκριμένους κανόνες συμπεριφοράς. Ένα αντικείμενο περιγράφεται από συγκεκριμένες ιδιότητες και κατά την εκτέλεση της εφαρμογής λαμβάνει μόνο τα μηνύματα που το αφορούν αγνοώντας οτιδήποτε άλλο συμβαίνει στο περιβάλλον του. Αυτό εξασφαλίζει ότι κάθε τμήμα του προγράμματος επηρεάζει μόνο τα αντικείμενα που το αφορούν και έχουμε πλήρη έλεγχο της εφαρμογής.

Ενθυλάκωση (Encapsulation). Όπως έχουμε ήδη αναφέρει, σε μια αντικειμενοστραφή εφαρμογή κάθε αντικείμενο αποτελεί ξεχωριστή οντότητα και περιέχει ενσωματωμένα τα χαρακτηριστικά (δεδομένα) και τους κανόνες συμπεριφοράς του (μεθόδους). Η δυνατότητα ενός αντικειμένου να συνδυάζει εσωτερικά τα δεδομένα και τις μεθόδους χειρισμού του καλείται ενθυλάκωση. Την ενθυλάκωση μπορούμε να την παρομοιάσουμε σαν ένα κέλυφος που υπάρχει γύρω από κάθε αντικείμενο και διαχωρίζει τον εσωτερικό από τον εξωτερικό του κόσμο. Με την ενθυλάκωση ο χειρισμός κάθε αντικειμένου είναι φυσικά πιο εύχρηστος και ασφαλής, γιατί τα περιεχόμενά του προστατεύονται και είναι δυνατό να μεταβληθούν μόνο με την αλλαγή των τιμών των ιδιοτήτων του ή με την δράση των μεθόδων που υποστηρίζει.

Κληρονομικότητα (Inheritance). Χαρακτηριστικά και μέθοδοι μπορούν να είναι κοινά σε διαφορετικές κλάσεις που είναι ιεραρχικά συνδεδεμένες. Μια κλάση μπορεί να περιγραφεί γενικά και στη συνέχεια μέσω αυτής της κλάσης να οριστούν υποκλάσεις (subclasses) αντικειμένων. Η κλάση απόγονος (υποκλάση) κληρονομεί και μπορεί να χρησιμοποιήσει όλα τα δεδομένα και τις μεθόδους που περιέχει η κλάση πρόγονος. Για παράδειγμα αν περιγράψουμε μια γενική κλάση γεωμετρικό σχήμα, μπορούμε στη συνέχεια να δημιουργήσουμε από αυτή υποκλάσεις αντικειμένων τριγώνου, τετραγώνου, κ.λπ. Η δυνατότητα δημιουργίας ιεραρχιών αντικειμένων καλείται κληρονομικότητα.

Πολυμορφισμός (Polymorphism). Με το χαρακτηριστικό του πολυμορφισμού παρέχεται η δυνατότητα στο ίδιο αντικείμενο να αναφέρεται, στο επίπεδο εκτέλεσης της εφαρμογής, σε διαφορετικές κλάσεις, να επιδρά διαφορετικά σε διαφορετικά αντικείμενα. Δύο ή περισσότερες κλάσεις αντικειμένων μπορούν να υποστηρίζουν συμπεριφορές με κοινό όνομα και ίδιο βασικό σκοπό αλλά με διαφορετική εφαρμογή. Για παράδειγμα σε ένα αυτοκίνητο τα πεντάλ του γκαζιού και του φρένου υποστηρίζουν τη μέθοδο πάτησε. Με τη μέθοδο πάτησε ο οδηγός δίνει μια εντολή για την εκτέλεση μιας εργασίας, αλλά η εφαρμογή της μεθόδου είναι διαφορετική σε κάθε πεντάλ. Όταν ο οδηγός πατήσει το γκάζι, το αυτοκίνητο αναπτύσσει ταχύτητα, ενώ το πάτημα του φρένου το επιβραδύνει. Η ίδια συμπεριφορά του οδηγού επιφέρει διαφορετικό αποτέλεσμα. Με τη χρήση του πολυμορφισμού ο προγραμματιστής απαλλάσσεται από τη σύνταξη και την επανάληψη πολύπλοκων δομών ελέγχου μέσα στην εφαρμογή.

παρέχει το περιβάλλον ανάπτυξης, πρέπει να επέμβουμε δυναμικά με τη χρήση εντολών κώδικα. Στα περισσότερα αντικειμενοστραφή προγραμματιστικά περιβάλλοντα, όπως η Visual C++ ή η Visual Basic, μια εντολή απόδοσης τιμής σε ιδιότητα έχει τη μορφή :

Αντικείμενο.Ιδιότητα = Τιμή

Στην παραπάνω εντολή ο όρος Αντικείμενο αντιπροσωπεύεται από μια ιδιότητα η οποία χαρακτηρίζει μοναδικά το επιθυμητό αντικείμενο. Συνήθως αυτή η ιδιότητα είναι το Όνομα του αντικειμένου.

11.1.4. Μέθοδοι

Σε πολλά αντικειμενοστραφή προγραμματιστικά περιβάλλοντα, όπως η Smalltalk και η Visual Basic, ένα αντικείμενο είναι δυνατόν εκτός από ιδιότητες να περιέχει και μεθόδους (methods). Μια μέθοδος είναι η εφαρμογή μιας ενέργειας επάνω σε ένα αντικείμενο. Οι μέθοδοι είναι εσωτερικά στοιχεία του αντικειμένου όπως και οι ιδιότητες.

Όλα τα αντικείμενα μιας κλάσης υποστηρίζουν τις ίδιες μεθόδους. Το αποτέλεσμα μιας μεθόδου επάνω σε ένα αντικείμενο συνήθως επηρεάζει κάποιες από τις τιμές των ιδιοτήτων του. Οι μέθοδοι μπορούν να ενεργήσουν σε ένα αντικείμενο μόνο κατά την εκτέλεση της εφαρμογής, άρα μέσα από εντολές κώδικα. Η σύνταξη μιας εντολής εκτέλεσης μεθόδου είναι :

Αντικείμενο.Μέθοδος

11.2. Οδηγούμενος από γεγονότα προγραμματισμός

Ο **οδηγούμενος από γεγονότα προγραμματισμός** (event-driven programming) είναι μια μεθοδολογία προγραμματισμού που μας επιτρέπει, σε συνδυασμό με τον αντικειμενοστραφή προγραμματισμό, να εκμεταλλευτούμε τα πλεονεκτήματα των σύγχρονων περιβαλλόντων εργασίας.

Για να μπορέσουμε να κατανοήσουμε καλύτερα τα πλεονεκτήματα του οδηγούμενου από γεγονότα προγραμματισμού, θα κάνουμε πρώτα μια σύγκρισή του με το δομημένο προγραμματισμό, που αποτελεί την παραδοσιακή μορφή προγραμματισμού.

Ένα εξωτερικό μήνυμα είναι δυνατό να προκαλέσει περισσότερα του ενός γεγονότων στην εφαρμογή.

Σε μια εφαρμογή που έχει αναπτυχθεί με τη φιλοσοφία του δομημένου προγραμματισμού, η εκτέλεσή της ξεκινά από την αρχική εντολή του προγράμματος και η ροή εκτέλεσής της είναι καθορισμένη από τις διαδικασίες και τις συναρτήσεις που περιλαμβάνει το πρόγραμμα. Σύμφωνα με τα παραπάνω το δομικό στοιχείο του προγράμματος αποτελούν οι διαδικασίες και οι συναρτήσεις. Σε όλη τη διάρκεια της εκτέλεση της εφαρμογής, το πρόγραμμα διατηρεί τον έλεγχό της, ενώ στο χρήστη έχει ανατεθεί δευτερεύων ρόλος και απλά πληκτρολογεί κάποια δεδομένα, όταν το πρόγραμμα το απαιτεί.

Αντίθετα με τον οδηγούμενο από γεγονότα προγραμματισμό, ο χρήστης με την έναρξη της εκτέλεσης της εφαρμογής αποκτά τον έλεγχό της και αποφασίζει, χρησιμοποιώντας το πληκτρολόγιο ή το ποντίκι, ποιο τμήμα του προγράμματος θα εκτελεστεί. Κάθε ενέργεια του χρήστη δημιουργεί μηνύματα με τη μορφή γεγονότων, τα οποία αντιλαμβάνεται το πρόγραμμα και ανταποκρίνεται σε αυτά.

Γεγονός (event) είναι μια ενέργεια που αναγνωρίζεται από την εφαρμογή, και συγκεκριμένα από τα αντικείμενά της, και την αναγκάζει να ανταποκριθεί. Η πρόκληση ενός γεγονότος μπορεί να γίνει είτε από το χρήστη με τη χρήση του πληκτρολόγιου ή του ποντικιού, είτε από το σύστημα.

Παραδείγματα γεγονότων που προκαλούνται από το χρήστη είναι το πάτημα κάποιου πλήκτρου εντολής, η επιλογή ενός στοιχείου μίας λίστας, η πληκτρολόγηση κάποιων χαρακτήρων σε ένα πλαίσιο κειμένου, η επιλογή ενός αντικειμένου μενού επιλογών κ.λπ. Ως γεγονότα που προκαλούνται από το σύστημα μπορούμε να αναφέρουμε την αποστολή μηνυμάτων από το ρολόι του υπολογιστή ή από άλλες εφαρμογές και ο τερματισμός του συστήματος.

11.2.1. Διαδικασίες

Σε πολλά από τα σύγχρονα προγραμματιστικά περιβάλλοντα, ο τρόπος που ο προγραμματιστής καθορίζει την εκτέλεση του προγράμματος, είναι με συγγραφή των κατάλληλων εντολών κώδικα που ενεργοποιούν τα αντικείμενα της εφαρμογής και τα υποχρεώνουν να αντιδράσουν στα γεγονότα που προκαλούνται στο περιβάλλον τους. Ο κώδικας για κάθε γεγονός που πρόκειται να συμβεί, σε άλλα προγραμματιστικά περιβάλλοντα γράφεται με τη μορφή ανεξάρτητων τμημάτων κώδικα του προγράμματος, που καλούνται **διαδικασίες γεγονότων** (event procedures), ενώ σε κάποια άλλα προγραμματιστικά περιβάλλοντα γράφεται ενιαία.

Κάθε αντικείμενο της εφαρμογής αναγνωρίζει ένα συγκεκριμένο αριθμό γεγονότων. Κατά συνέπεια, για εκείνη την κατηγορία των προγραμματιστικών περιβαλλόντων που υποστηρίζουν διαδικασίες γεγονότων, αυτές καθορίζονται από την ενέργεια και το αντικείμενο που τις υποστηρίζει.

Η σύνταξη μιας διαδικασίας γεγονότος είναι :

```
ΔΙΑΔΙΚΑΣΙΑ Αντικείμενο_Γεγονός
    ...
Εντολές κώδικα
    ...
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ
```

Το μεγάλο πλεονέκτημα των σύγχρονων περιβαλλόντων προγραμματισμού, είναι ότι κάθε αντικείμενο αναγνωρίζει αυτόματα τα γεγονότα που το αφορούν και ξέρει πως θα αντιδράσει σε αυτά. Ο προγραμματιστής σε κανένα σημείο του προγράμματος δεν χρειάζεται να ασχοληθεί με τον τρόπο που το αντικείμενο θα ανιχνεύσει από το περιβάλλον το γεγονός, ούτε με την κλήση του αντίστοιχου κατάλληλου κώδικα, η οποία θα γίνει αυτόματα από το αντικείμενο.

Η έννοια της διαδικασίας δεν απαντάται με την ίδια ονομασία σε όλα τα σύγχρονα προγραμματιστικά περιβάλλοντα. Για παράδειγμα, στη Visual C++ οι διαδικασίες ονομάζονται συναρτήσεις (functions).

Σε μια εφαρμογή οδηγούμενη από τα γεγονότα δεν συμπεριλαμβάνουμε μόνο διαδικασίες γεγονότων, αλλά έχουμε τη δυνατότητα να συμπεριλάβουμε και **γενικές διαδικασίες** ή **συναρτήσεις** που καλούνται μέσα από διαδικασίες γεγονότων.

Για παράδειγμα αν θέλουμε σε πολλά σημεία ενός προγράμματος να γίνεται ο ίδιος υπολογισμός, αποφεύγουμε να χρησιμοποιούμε τις ίδιες εντολές προγράμματος σε κάθε διαδικασία γεγονότος, αλλά δημιουργούμε μια γενική διαδικασία *Υπολογισμός* εφαρμόζοντας τις αρχές του δομημένου προγραμματισμού :

```
ΔΙΑΔΙΚΑΣΙΑ Υπολογισμός
    ...
    Εντολές κώδικα
    ...
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ
```

Στη συνέχεια μέσα από κάθε διαδικασία γεγονότος στο σημείο που θέλουμε να εκτελεστεί ο συγκεκριμένος υπολογισμός καλούμε τη γενική διαδικασία :

```
ΔΙΑΔΙΚΑΣΙΑ Αντικείμενο_Γεγονός
    ...
    Κάλεσε Υπολογισμός      ' Κλήση της γενικής διαδικασίας
    ...
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ
```

Παρατηρούμε λοιπόν ότι για την ανάπτυξη μιας σύγχρονης εφαρμογής πρέπει να εφαρμόσουμε και τις τρεις τεχνικές προγραμματισμού που έχουμε αναπτύξει. Σε μια σύγχρονη εφαρμογή συνδυάζουμε τον **τμηματικό** (modular) προγραμματισμό, αξιοποιώντας τις τεχνικές του αντικειμενοστραφούς και του οδηγούμενου από γεγονότα προγραμματισμού, αλλά εξακολουθούμε να χρησιμοποιούμε και το **δομημένο** (structural) προγραμματισμό, αφού τα περιεχόμενα των διαδικασίων γεγονότων είναι δομημένες εντολές κώδικα. Η εφαρμογή του δομημένου προγραμματισμού φυσικά δε διαφοροποιείται και ισχύουν όσα γνωρίζουμε για τις μεταβλητές, τις δομές ελέγχου, τους πίνακες κ.ά. Τέλος μέσα σε ένα πρόγραμμα έχουμε τη δυνατότητα χρησιμοποίησης γενικών διαδικασιών και συναρτήσεων που επίσης στηρίζονται στις αρχές του δομημένου προγραμματισμού.

11.2.2. Ροή εκτέλεσης εφαρμογής

Αν προσπαθήσουμε να αναλύσουμε τη ροή εκτέλεσης μιας εφαρμογής που ακολουθεί την τεχνική του οδηγούμενου από γεγονότα προγραμματισμού και υποστηρίζει διαδικασίες γεγονότων, μπορούμε να τη παρουσιάσουμε με τα παρακάτω βήματα :

1. Ο χρήστης ξεκινά την εκτέλεση της εφαρμογής.
2. Τα αντικείμενα της εφαρμογής περνούν σε κατάσταση αναμονής γεγονότων.
3. Μόλις ένα αντικείμενο αναγνωρίσει μέσα στο περιβάλλον του κάποιο γεγονός που το αφορά, καλεί την αντίστοιχη διαδικασία γεγονότος.

4. Αν στη διαδικασία γεγονότος έχουμε συμπεριλάβει εντολές κώδικα, εκτελούνται διαφορετικά η εφαρμογή αγνοεί το γεγονός.
5. Τα αντικείμενα της εφαρμογής περνούν πάλι σε κατάσταση αναμονής γεγονότων (Βήμα 2).

Ο προγραμματιστής δεν είναι απαραίτητο να συμπεριλάβει εντολές κώδικα σε κάθε διαδικασία γεγονότος που υποστηρίζουν τα αντικείμενα της εφαρμογής. Απλά επιλέγει αυτές για τις οποίες τον ενδιαφέρει το πρόγραμμα να αντιδρά. Αν μια διαδικασία γεγονότος δεν περιέχει καμία εντολή κώδικα, όπως είναι φυσικό αυτό το γεγονός αγνοείται τελείως από την εφαρμογή, σαν να μην συνέβη ποτέ.

Ένα γεγονός είναι δυνατό να προκληθεί και μέσα από εντολές κώδικα. Η επιτυχία μιας οδηγούμενης από γεγονότα εφαρμογής εξαρτάται από τον τρόπο που ο προγραμματιστής χρησιμοποιεί τα γεγονότα. Η πρόκληση γεγονότων μέσα από κώδικα αποτελεί μια έξυπνη τεχνική.

11.3. Υλοποίηση εφαρμογών σε σύγχρονο προγραμματιστικό περιβάλλον

Στο παρελθόν ένας προγραμματιστής κατά το σχεδιασμό της εφαρμογής έδινε ιδιαίτερη έμφαση στην ορθή υλοποίηση του αλγορίθμου του προγράμματος και δεν ασχολούνταν σχεδόν καθόλου με τον τρόπο επικοινωνίας του χρήστη με την εφαρμογή, αφού άλλωστε το περιβάλλον εργασίας δεν του παρείχε και ιδιαίτερες δυνατότητες επάνω σε αυτό το τμήμα του προγράμματος.

Αντιθέτως σε ένα σύγχρονο περιβάλλον εργασίας δίνεται ιδιαίτερη έμφαση στον τρόπο που επικοινωνεί ο χρήστης με το σύστημα. Κατά συνέπεια ο προγραμματιστής είναι υποχρεωμένος να ασχοληθεί ιδιαίτερα με τον τρόπο επικοινωνίας του χρήστη με την εφαρμογή, τη **διεπαφή χρήστη** (user interface). Κατά το σχεδιασμό της διεπαφής χρήστη, ο προγραμματιστής πρέπει να αντιλαμβάνεται την εφαρμογή από την πλευρά του χρήστη. Ένας προγραμματιστής για να δημιουργήσει μια σύγχρονη, φιλική και ευέλικτη εφαρμογή θα πρέπει να δώσει οπτική μορφή στο περιβάλλον εργασίας που παρέχει στο χρήστη. Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρέχουν στον προγραμματιστή όλα εκείνα τα εργαλεία που χρειάζεται για να δημιουργήσει ένα τέτοιο περιβάλλον. Και φυσικά τα εργαλεία αυτά είναι τα αντικείμενα.

Στον κόσμο των αντικειμένων, ένας προγραμματιστής είναι φυσικό να ξεκινά το σχεδιασμό της εφαρμογής από τα αντικείμενα που πρόκειται να χρησιμοποιήσει. Η σωστή επιλογή των αντικειμένων εξασφαλίζει σε μεγάλο βαθμό και την επιτυχία εκτέλεσης της εφαρμογής.

Μετά από την επιλογή των κατάλληλων αντικειμένων, ο προγραμματιστής γνωρίζει ποιες θα είναι οι πιθανές ενέργειες του χρήστη ή του συστήματος επί της εφαρμογής. Επομένως είναι ευδιάκριτο ποια είναι τα γεγονότα που είναι δυνατό να ανιχνεύσουν τα επιλεγμένα αντικείμενα. Από τα πιθανά γεγονότα που θα συμβούν στο περιβάλλον της εφαρμογής, ο προγραμματιστής επιλέγει αυτά που θέλει να αξιοποιήσει μέσα από την εφαρμογή του και γράφει τον κατάλληλο κώδικα.

Συνοψίζοντας τα παραπάνω μπορούμε να περιγράψουμε την ανάπτυξη μιας εφαρμογής σε ένα σύγχρονο προγραμματιστικό περιβάλλον σαν διαδικασία τριών βημάτων :

1. Σχεδιασμός του τρόπου επικοινωνίας χρήστη-εφαρμογής, επιλέγοντας τα κατάλληλα αντικείμενα.
2. Καθορισμός της αρχικής συμπεριφοράς των αντικειμένων μέσω των ιδιοτήτων που υποστηρίζουν.
3. Δημιουργία και εκσφαλμάτωση κώδικα.

Παράδειγμα

Για να γίνει κατανοητός ο τρόπος υλοποίησης μιας εφαρμογής σε ένα σύγχρονο προγραμματιστικό περιβάλλον, θα τον περιγράψουμε, θεωρώντας ένα υποθετικό προγραμματιστικό περιβάλλον, παρουσιάζοντας ένα απλό παράδειγμα και περιγράφοντας αναλυτικά όλα τα βήματα σχεδιασμού.

Το πρόβλημα συνίσταται στη δημιουργία μιας εφαρμογής που θα διαθέτει πλήκτρα, το πάτημα των οποίων θα έχει σαν αποτέλεσμα να εμφανίζεται καθένα από τα τρία παραπάνω περιγραφόμενα βήματα σχεδιασμού μιας εφαρμογής σε ένα σύγχρονο προγραμματιστικό περιβάλλον.

Δημιουργία της διεπαφής χρήστη

Το πρώτο στοιχείο που πρέπει να απασχολήσει ένα προγραμματιστή, είναι ο τρόπος επικοινωνίας του χρήστη με την εφαρμογή. Για το σχεδιασμό λοιπόν της διεπαφής χρήστη πρέπει να επιλέξουμε τα κατάλληλα αντικείμενα. Στο παράδειγμά μας θα χρησιμοποιήσουμε :

- ένα αντικείμενο **φόρμα** το οποίο είναι το παράθυρο μέσα στο οποίο θα εκτελείται η εφαρμογή,
- αντικείμενα **πλήκτρα εντολής** με τα οποία ο χρήστης θα καθοδηγεί την εκτέλεσή της ,

- αντικείμενα **ετικέτες** που θα εμφανίζουν τις πληροφορίες.

Αρχικά δημιουργούμε μια φόρμα και στη συνέχεια τοποθετούμε επάνω της τρία πλήκτρα εντολής. Τα τρία πλήκτρα εντολής τα επιλέγουμε από την εργαλειοθήκη (toolbox) που μας παρέχει το υποθετικό περιβάλλον προγραμματισμού στο οποίο βρισκόμαστε.

Τα τρία πλήκτρα εντολής θα δίνουν τη δυνατότητα στο χρήστη να επιλέγει το βήμα της περιγραφής ανάπτυξης μιας εφαρμογής σε ένα σύγχρονο προγραμματιστικό περιβάλλον, που θέλει να εμφανιστεί. Για να είναι τα πλήκτρα κατατοπιστικά θα εμφανίσουμε επάνω σε κάθε ένα τον αριθμό του βήματος που πρόκειται να παρουσιάζεται. Ακόμη πρέπει να αποδώσουμε σε καθένα από τα πλήκτρα ένα μοναδικό όνομα, με το οποίο θα επικοινωνούμε μαζί του μέσα από το κώδικα της εφαρμογής. Η μετατροπή αυτών των χαρακτηριστικών μπορεί να γίνει αλλάζοντας τις τιμές ιδιοτήτων των αντικειμένων.

Για το υποθετικό σύγχρονο προγραμματιστικό περιβάλλον που βρισκόμαστε, κάνουμε τις παρακάτω παραδοχές :

- η απόδοση τιμών στις ιδιότητες των αντικειμένων γίνεται με χρήση κατάλληλου εργαλείου (παράθυρο ιδιοτήτων) που διαθέτει,
- η απόδοση προγραμματιστικού ονόματος σε ένα αντικείμενο πλήκτρο εντολής γίνεται μέσω της ιδιότητάς του **Όνομα**,
- η απόδοση λεκτικού πάνω σε ένα αντικείμενο πλήκτρο εντολής γίνεται μέσω της ιδιότητάς του **Τίτλος**,
- η απόδοση γραφικού πάνω σε ένα αντικείμενο πλήκτρο εντολής γίνεται μέσω της ιδιότητάς του **Εικόνα**,

Για τα τρία πλήκτρα εντολής πρέπει να αντιστοιχήσουμε τις παρακάτω τιμές στις προαναφερθείσες ιδιότητες :

Αντικείμενο	Ιδιότητα	Τιμή
Πλήκτρο εντολής 1	Όνομα	ΠλήκτροΕντολήςΒήμα1
	Τίτλος	1
Πλήκτρο εντολής 2	Όνομα	ΠλήκτροΕντολήςΒήμα2
	Τίτλος	2
Πλήκτρο εντολής 3	Όνομα	ΠλήκτροΕντολήςΒήμα3
	Τίτλος	3

Η τιμή στην ιδιότητα Όνομα, είναι σωστό να προσδιορίζει τη κλάση από την οποία προέρχεται το αντικείμενο και το ρόλο του μέσα στην εφαρμογή. Για παράδειγμα με την τιμή ΠλήκτροΕντολήςΒήμα1, γίνεται αντιληπτό ότι αναφερόμαστε σε ένα πλήκτρο εντολής, που εμφανίζει το πρώτο βήμα.

Στη συνέχεια ακολουθώντας την ίδια διαδικασία, τοποθετούμε τέσσερα αντικείμενα Ετικέτα επάνω στη φόρμα. Η πρώτη ετικέτα θα περιγράφει το σκοπό της εφαρμογής και οι τρεις επόμενες τα βήματα σχεδιασμού της εφαρμογής. Τέλος στην εφαρμογή θα τοποθετήσουμε ένα ακόμη πλήκτρο εντολής που θα του αποδώσουμε γραφική μορφή, και με το οποίο θα τερματίζεται η εκτέλεση της εφαρμογής.

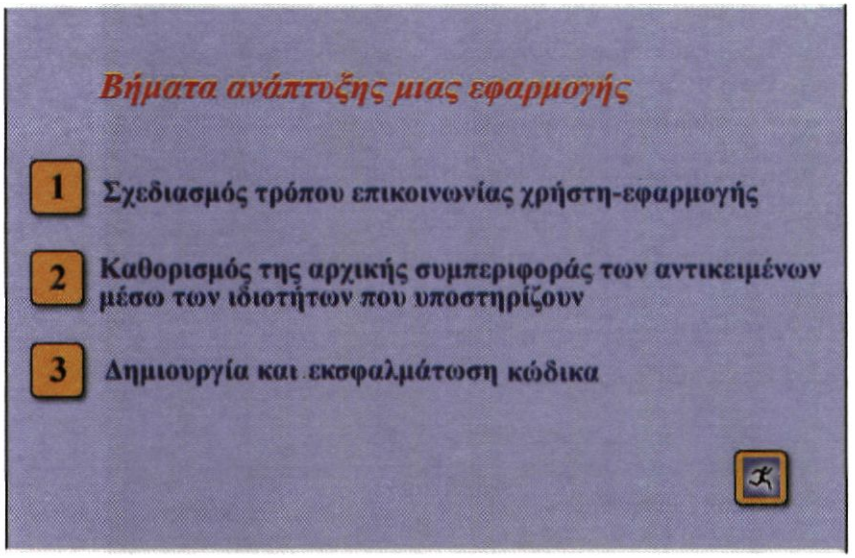
Για τα εργαλεία Ετικέτα στο υποθετικό προγραμματιστικό περιβάλλον που βρισκόμαστε, κάνουμε τις εξής παραδοχές :

- έχουν τη δυνατότητα να εμφανίζουν κείμενο που καταχωρείται κατά τη διάρκεια σχεδίασης της εφαρμογής, αλλά ο χρήστης δεν έχει τη δυνατότητα να το τροποποιήσει κατά τη διάρκεια εκτέλεσης της εφαρμογής,
- η καταχώριση κειμένου σε ένα αντικείμενο ετικέτα γίνεται μέσω της ιδιότητάς της **Τίτλος**,
- η απόδοση προγραμματιστικού ονόματος σε ένα αντικείμενο ετικέτα γίνεται μέσω της ιδιότητάς της **Όνομα**,
- η εμφάνιση ή όχι του κείμενου που περιλαμβάνει ένα αντικείμενο ετικέτα εξαρτάται από την τιμή που έχει η ιδιότητά του **Ορατό** και η οποία δέχεται τις τιμές Αληθής ή Ψευδής.

Στα υπόλοιπα αντικείμενα της εφαρμογής θα αποδώσουμε τις παρακάτω τιμές :

Αντικείμενο	Ιδιότητα	Τιμή
Ετικέτα 0	Όνομα	ΕτικέταΤίτλος
	Ορατό	Αληθής
	Τίτλος	Βήματα ανάπτυξης μιας εφαρμογής
Ετικέτα 1	Όνομα	ΕτικέταΒήμα1
	Ορατό	Ψευδής
	Τίτλος	Σχεδιασμός του τρόπου επικοινωνίας χρήστη - εφαρμογής, επιλέγοντας τα κατάλληλα αντικείμενα
Ετικέτα 2	Όνομα	ΕτικέταΒήμα2
	Ορατό	Ψευδής
	Τίτλος	Καθορισμός της αρχικής συμπεριφοράς των αντικειμένων μέσω των ιδιοτήτων που τα χαρακτηρίζουν

Αντικείμενο	Ιδιότητα	Τιμή
Ετικέτα 3	Όνομα	ΕτικέταΒήμα3
	Ορατό	Ψευδής
	Τίτλος	Δημιουργία και εκσφαλμάτωση κώδικα
Πλήκτρο εντολής 4	Όνομα	ΠλήκτροΕντολήςΤέλος
	Εικόνα	(εικονίδιο εξόδου)



Σχ. 11.4. Η διασύνδεση του χρήστη με την εφαρμογή στο παράδειγμά μας

Σε κάθε παράθυρο μιας εφαρμογής πρέπει να δίνεται στο χρήστη ένας εμφανής τρόπος εξόδου από την εφαρμογή, όπως γίνεται στο παράδειγμά μας με το πλήκτρο εντολής ΠλήκτροΕντολήςΤέλος. Με αυτή τη τεχνική δημιουργούμε φιλικές εφαρμογές και δίνουμε τη δυνατότητα χειρισμού τους ακόμη και σε αρχάριους χρήστες.

Μετά από την αντιστοίχηση όλων των ιδιοτήτων έχουμε ολοκληρώσει τη σχεδίαση της διασύνδεσης του χρήστη με την εφαρμογή, έτσι όπως φαίνεται στο σχήμα 11.4.

Είναι χαρακτηριστικό ότι στο υποθετικό σύγχρονο προγραμματιστικό περιβάλλον που βρισκόμαστε, η υλοποίηση της επικοινωνίας του χρήστη με την εφαρμογή έχει ολοκληρωθεί χωρίς να έχουμε γράψει ούτε μια εντολή κώδικα. Απλά έχουμε επιλέξει τα αντικείμενα και αποδώσαμε τις κατάλληλες τιμές σε ορισμένες από τις ιδιότητές τους. Παρόμοια όμως συμβαίνει και σε πολλά πραγματικά προγραμματιστικά περιβάλλοντα.

Ο κώδικας

Το τελευταίο βήμα που πρέπει να εκτελέσουμε για να ολοκληρωθεί ο σχεδιασμός της εφαρμογής, είναι η προσθήκη των εντολών κώδικα. Μέσα από τις εντολές κώδικα θα εκτελούμε τις εργασίες της εφαρμογής.

Στο παράδειγμά μας όλες οι παρεμβάσεις στην εκτέλεση της εφαρμογής, από την πλευρά του χρήστη, γίνονται μέσω των πλήκτρων εντολής.

Στο υποθετικό σύγχρονο προγραμματιστικό περιβάλλον που βρισκόμαστε κάνουμε την εξής παραδοχή αναφορικά με τα προκαλούμενα γεγονότα από τις ενέργειες του χρήστη :

- Το γεγονός που προκαλείται μόλις ο χρήστης πατήσει το αριστερό πλήκτρο του ποντικιού, όταν ο δείκτης του ποντικιού βρίσκεται πάνω από κάποιο πλήκτρο εντολής, είναι το γεγονός **Κλικ**.

Επομένως ο κώδικας της εφαρμογής πρέπει να γραφτεί στις διαδικασίες γεγονότων Κλικ των αντικειμένων πλήκτρα εντολής.

Το πάτημα κάθε πλήκτρου εντολής εμφάνισης των βημάτων, πρέπει κάθε φορά να εμφανίζει το μήνυμα που αντιστοιχεί σε αυτό και να αποκρύπτει τις υπόλοιπες ετικέτες βημάτων.

Ο κώδικας των τριών πλήκτρων εντολής λοιπόν θα είναι :

```
ΔΙΑΔΙΚΑΣΙΑ ΠλήκτροΕντολήςΒήμα1_Κλικ()
    ΕτικέταΒήμα1.Όρατο = Αληθής
    ΕτικέταΒήμα2.Όρατο = Ψευδής
    ΕτικέταΒήμα3.Όρατο = Ψευδής
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ

ΔΙΑΔΙΚΑΣΙΑ ΠλήκτροΕντολήςΒήμα2_Κλικ()
    ΕτικέταΒήμα1.Όρατο = Ψευδής
    ΕτικέταΒήμα2.Όρατο = Αληθής
    ΕτικέταΒήμα3.Όρατο = Ψευδής
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ

ΔΙΑΔΙΚΑΣΙΑ ΠλήκτροΕντολήςΒήμα3_Κλικ()
    ΕτικέταΒήμα1.Όρατο = Ψευδής
    ΕτικέταΒήμα2.Όρατο = Ψευδής
    ΕτικέταΒήμα3.Όρατο = Αληθής
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ
```

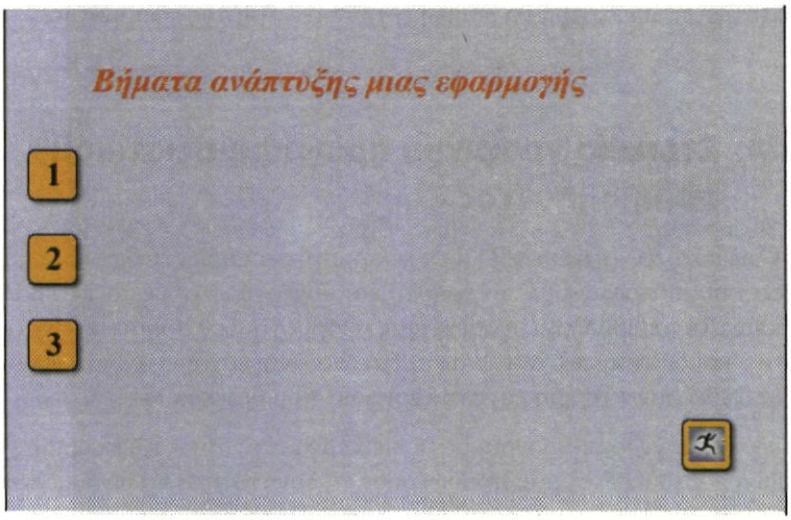
Τέλος στην διαδικασία γεγονότος Κλικ του πλήκτρου τερματισμού ΠλήκτροΕντολήςΤέλος της εφαρμογής πρέπει να συμπεριλάβουμε την εντολή με την οποία τερματίζεται η εκτέλεση της εφαρμογής :

```
ΔΙΑΔΙΚΑΣΙΑ ΠλήκτροΕντολήςΤέλος_Κλικ()
    Εντολή τερματισμού εφαρμογής
ΤΕΛΟΣ ΔΙΑΔΙΚΑΣΙΑΣ
```

Στο παράδειγμά μας, χρησιμοποιήσαμε μόνο το γεγονός Κλικ, για λόγους απλότητας. Η εφαρμογή αν υλοποιούταν σε πραγματικό προγραμματιστικό περιβάλλον θα παρουσίαζε πολλά κοινά σημεία με αυτά που παρουσιάστηκαν στο υποθετικό. Θα είχε βέβαια και πολλά περιθώρια βελτίωσης και πολλούς εναλλακτικούς τρόπους προσέγγισης, αλλά σκοπός είναι η εισαγωγή στη φιλοσοφία και στη μεθοδολογία προγραμματισμού και όχι η επίδειξη δυνατοτήτων ενός σύγχρονου, έστω και υποθετικού, προγραμματιστικού περιβάλλοντος.

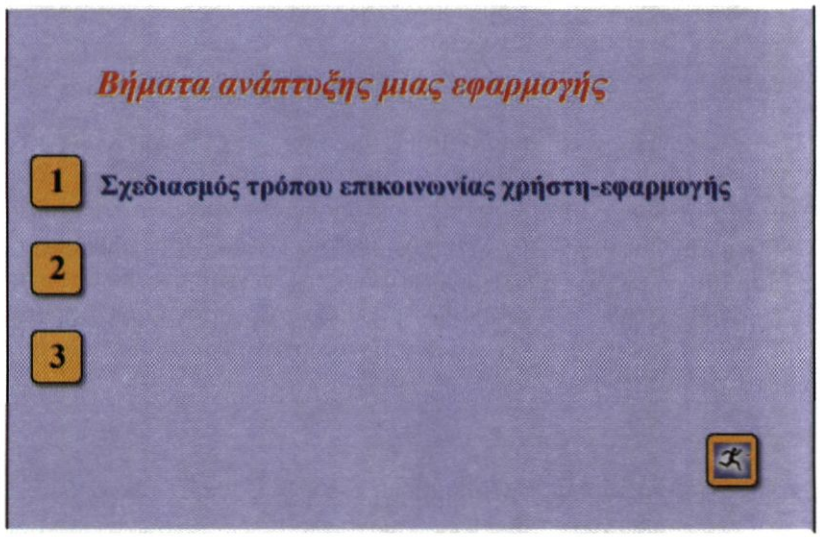
Η εκτέλεση

Μόλις ολοκληρώσουμε την προσθήκη του κώδικα στις διαδικασίες γεγονότων που θέλουμε να αντιδρά η εφαρμογή, είμαστε σε θέση να εκκινήσουμε την εκτέλεσή της στο υποθετικό προγραμματιστικό περιβάλλον μας. Μόλις ξεκινήσει η εφαρμογή θα εμφανίζεται η αρχική της οθόνη (σχήμα 11.5).



Σχ. 11.5. *Η πρώτη οθόνη του παραδείγματος σε μορφή εκτέλεσης*

Στη συνέχεια ο χρήστης έχει τη δυνατότητα πατώντας κάποιο από τα τρία αριθμημένα πλήκτρα να εμφανίσει κάποιο από τα τρία βήματα σχεδιασμού της εφαρμογής (σχήμα 11.6).



Σχ. 11.6. Ένα από τα τρία βήματα εκτέλεσης της εφαρμογής

Χρησιμοποιώντας το πλήκτρο εντολής με το γραφικό, ο χρήστης έχει τη δυνατότητα κάθε στιγμή να διακόψει την εκτέλεση της εφαρμογής.

11.4. Στοιχεία γραφικού προγραμματιστικού περιβάλλοντος

Η φιλικότητα και η ευελιξία μιας εφαρμογής καθορίζεται κυρίως από το τρόπο που επικοινωνεί με το χρήστη και συνεργάζεται με το περιβάλλον εργασίας. Το περιβάλλον εργασίας μιας εφαρμογής δεν πρέπει να είναι κουραστικό και δύσχρηστο. Αντίθετα πρέπει να είναι ευχάριστο, να καθοδηγεί και να διευκολύνει το χρήστη στην εργασία που θέλει να επιτελέσει.

Όλα τα σύγχρονα προγραμματιστικά εργαλεία, όπως η Visual C++, το Delphi ή η Visual Basic, μας προσφέρουν τη δυνατότητα να δίνουμε στις εφαρμογές μας τέτοια χαρακτηριστικά μέσα από τη χρησιμοποίηση των γραφικών. Με τον προγραμματισμό σε γραφικό προγραμματιστικό περιβάλλον, παρέχοντας ένα εύχρηστο περιβάλλον εργασίας στους χρήστες, αυξάνουμε τις δυνατότητές τους και μεγιστοποιούμε τα αποτελέσματα. Επιπλέον, μειώνεται δραστικά ο χρόνος και το κόστος εκμάθησης των χρηστών σε μια καινούργια εφαρμογή.

Ένα γραφικό προγραμματιστικό περιβάλλον μας δίνει ευελιξία και διαφορετικές δυνατότητες δημιουργίας γραφικού τρόπου επικοινωνίας του χρήστη με την εφαρμογή, αλλά ταυτόχρονα μας καθοδηγεί να δημιουργήσουμε γνώριμες εφαρμογές, αφού τα αντικείμενα, όπως πλήκτρα εντολής, λίστες, πλαίσια ελέγχου, πλήκτρα επιλογής, που χρησιμοποιούμε στις δικές μας εφαρμογές είναι ήδη γνωστά στους χρήστες από εφαρμογές γενικής χρήσης. Έτσι οι εφαρμογές μας, διατηρούν κάποια φιλοσοφία που είναι ήδη γνωστή στους χρήστες από άλλες εφαρμογές, όπως λογιστικά φύλλα, επεξεργαστές κειμένου ή βάσεις δεδομένων, που χρησιμοποιούν παρόμοια γραφικά αντικείμενα.

Ακόμα, ένα γραφικό προγραμματιστικό περιβάλλον προσφέρει στον προγραμματιστή τα κατάλληλα εργαλεία για γρήγορη ανάπτυξη εφαρμογών. Έτοιμα εργαλεία που παρέχονται από πολλά προγραμματιστικά περιβάλλοντα, όπως εργαλειοθήκες (toolboxes), εργαλεία εκσφαλμάτωσης (debugging tools), εύχρηστα παράθυρα συγγραφής κώδικα (code windows), γεννήτριες εκτυπωτικών αναφορών (report generators) συνθέτουν ένα περιβάλλον που διευκολύνει τον προγραμματιστή στο έργο του. Ακόμη διάφοροι οδηγοί εκτέλεσης εργασιών (wizards) τον βοηθούν στην εκτέλεση συχνών εργασιών, όπως τη δημιουργία φορμών ή εκτυπώσεων. Τέλος ένα σύγχρονο προγραμματιστικό εργαλείο, επιτρέπει στον προγραμματιστή να εκμεταλλευτεί τις ήδη υπάρχουσες εφαρμογές του συστήματος, χωρίς να χρειάζεται να δημιουργεί τμήματα κώδικα στην εφαρμογή του για την εκτέλεση εργασιών που ήδη εκτελεί σωστά κάποια άλλη εφαρμογή.

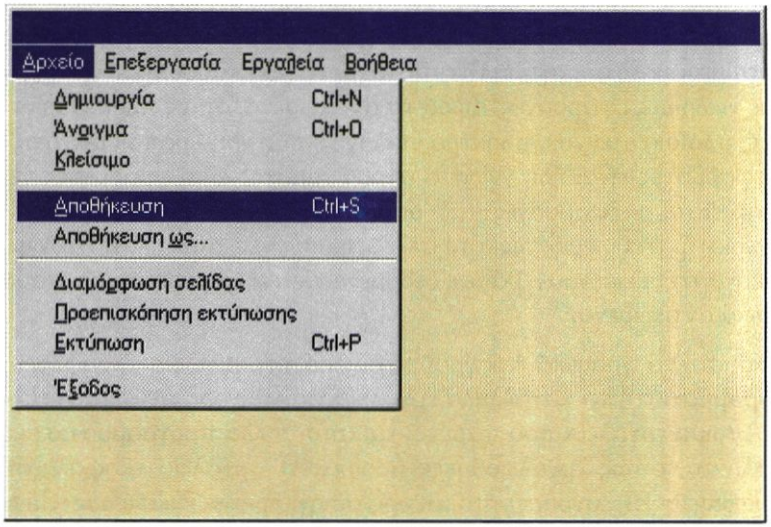
Το αποτέλεσμα εργασίας με ένα γραφικό προγραμματιστικό εργαλείο, είναι η γρήγορη υλοποίηση δυναμικών εφαρμογών που επικοινωνούν φιλικά με το χρήστη και συνεργάζονται αρμονικά με το περιβάλλον και τις υπόλοιπες εφαρμογές.

Βασικά χαρακτηριστικά στοιχεία των γραφικών προγραμματιστικών περιβαλλόντων είναι τα μενού επιλογών και τα πλαίσια διαλόγου.

11.4.1 Μενού επιλογών

Τα **μενού επιλογών** (menus) μας επιτρέπουν, κατά την εκτέλεση μιας εφαρμογής, τη γρήγορη πρόσβαση σε μια εύκολα προσπελάσιμη λίστα επιλογών. Οι περισσότεροι χρήστες γνωρίζουν ήδη τις δυνατότητες των μενού επιλογών αφού τα έχουν ήδη χρησιμοποιήσει μέσα από γνωστές εφαρμογές γενικής χρήσης.

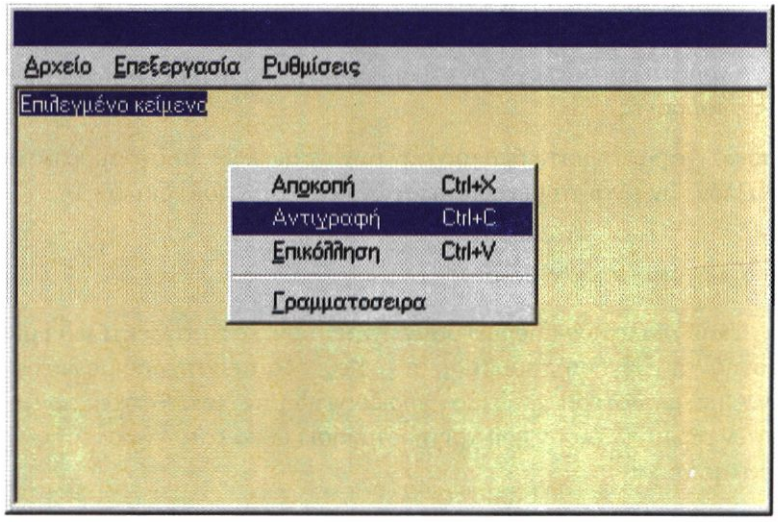
Τα κλασικά μενού επιλογών εμφανίζονται στη **γραμμή μενού** (menu bar) ενός παραθύρου (σχήμα 11.7).



Σχ. 11.7. Ένα κλασικό μενού επιλογών

Η υλοποίηση ενός μενού επιλογών σε αρκετά από τα σύγχρονα προγραμματιστικά περιβάλλοντα γίνεται μέσα από σχεδιαστικά εργαλεία που προσφέρουν, τους Επεξεργαστές μενού επιλογών (Menu Editor).

Πολλά περιβάλλοντα παρέχουν τη δυνατότητα εμφάνισης μενού επιλογών με τη μορφή **πτυσσόμενων μενού** (popup menu) (σχήμα 11.8). Τα πτυσσόμενα μενού επιλογών, συνήθως εμφανίζονται με το πάτημα του δεξιού πλήκτρου του ποντικιού και τα χρησιμοποιούμε για τη εμφάνιση σύντομων επιλογών σε συγκεκριμένα σημεία της εφαρμογής.



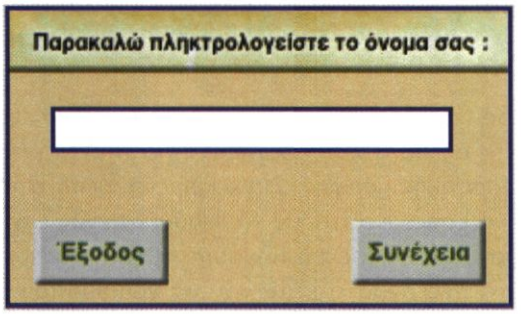
Σχ. 11.8. Ένα πτυσσόμενο μενού επιλογών

Σχ. 11.9. Μια γραμμή εργαλείων

Στις εφαρμογές μας αν θέλουμε να δώσουμε ένα σύντομο τρόπο πρόσβασης στις πιο συχνά εκτελέσιμες επιλογές ενός μενού, ο καλύτερος τρόπος είναι η δημιουργία **γραμμών εργαλείων** (toolbars) (σχήμα 11.9). Μια γραμμή εργαλείων μπορεί να υλοποιηθεί πολύ εύκολα με την ομαδοποίηση αντικειμένων εικόνας ή πλήκτρων εντολής που τα τοποθετούμε κάτω από τη γραμμή μενού και τον αντίστοιχο κώδικα.

11.4.2 Πλαίσια διαλόγου

Τα **πλαίσια διαλόγου** (dialog boxes) (σχήμα 11.10) εξασφαλίζουν ένα άμεσο διάλογο της εφαρμογής με το χρήστη. Πολλές φορές κατά την εκτέλεση μιας εφαρμογής είναι αναγκαίο ο χρήστης να επικοινωνεί με την εφαρμογή. Μέσα από τα πλαίσια διαλόγου είναι δυνατό να εμφανιστούν μηνύματα προς το χρήστη, αλλά και ο χρήστης μπορεί να εισάγει κάποια δεδομένα στην εφαρμογή.



Σχ. 11.10. Ένα πλαίσιο διαλόγου

Η σχεδίαση ενός πλαίσιου διαλόγου πρέπει να αντιμετωπίζεται από την πλευρά του χρήστη. Πρέπει να χρησιμοποιούνται κατανοητά μηνύματα και να εμφανίζονται οι κατάλληλες επιλογές για την περίσταση.

Τα περισσότερα σύγχρονα προγραμματιστικά εργαλεία παρέχουν στο προγραμματιστή έτοιμα προκαθορισμένα πλαίσια διαλόγου που μπορεί να χρησιμοποιήσει στις εφαρμογές του. Τα προκαθορισμένα πλαίσια διαλόγου, έχουν το πλεονέκτημα ότι διατηρούν την ίδια φιλοσοφία με το περιβάλλον εργασίας, αφού είναι πανομοιότυπα με τα πλαίσια που εμφανίζει το σύστημα όταν πρέπει να επικοινωνήσει με το χρήστη, για παράδειγμα σε περίπτωση λάθους. Σε ένα προκαθορισμένο πλαίσιο διαλόγου έχουμε τη δυνατότητα να εμφανίσουμε διάφορα πλήκτρα, όπως πλήκτρο επικύρω-

σης (OK), ακύρωσης (Cancel) και επανάληψης (Retry). Η εμφάνιση των προκαθορισμένων πλαισίων διαλόγου γίνεται μέσω συναρτήσεων της γλώσσας προγραμματισμού.

Αν κανένα από τα προκαθορισμένα πλαίσια διαλόγου, που μας προσφέρει το περιβάλλον προγραμματισμού δεν καλύπτει κάποια ανάγκη της εφαρμογής μας ή δεν μας εκφράζει η τυποποίησή τους, έχουμε φυσικά τη δυνατότητα τοποθετώντας αντικείμενα επάνω σε μια φόρμα να δημιουργήσουμε τα δικά μας πλαίσια διαλόγου.

11.5. Επικοινωνία με άλλες εφαρμογές

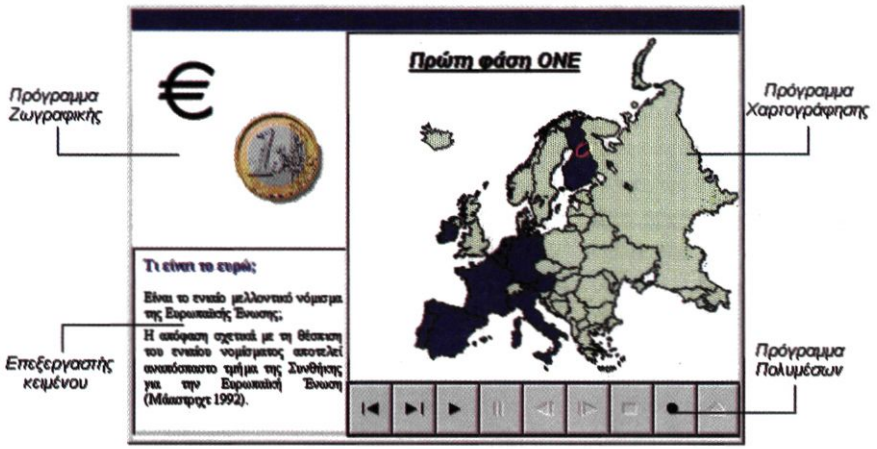
Οι προγραμματιστές στον παραδοσιακό προγραμματισμό τις περισσότερες φορές αγνοούσαν υπάρχουσες εφαρμογές, γιατί το περιβάλλον εργασίας καθιστούσε δύσκολη τη συνεργασία τους. Έτσι κατέφευγαν στη συγγραφή τμημάτων κώδικα για την εκτέλεση εργασιών, που ήδη εκτελούσε ορθά μια άλλη εφαρμογή.

Όμως τα σύγχρονα περιβάλλοντα εργασίας παρέχουν τους μηχανισμούς για υψηλού επιπέδου επικοινωνία μεταξύ διαφορετικών εφαρμογών, του λειτουργικού συστήματος και της εφαρμογής μας και μας επιτρέπουν έτσι την εκμετάλλευση έτοιμων βιβλιοθηκών προγραμμάτων, ελαττώνοντας και τις απαιτήσεις σε συγγραφή κώδικα.

Ο προγραμματιστής σε ένα σύγχρονο προγραμματιστικό περιβάλλον, αντιμετωπίζει κάθε ολοκληρωμένο πρόβλημα σαν μια ανεξάρτητη εφαρμογή. Αν κάποια από τις εφαρμογές που ήδη κατέχει μπορεί να επιλύσει το πρόβλημα, τότε έχει την ευχέρεια να τη χρησιμοποιήσει. Αυτός ο τρόπος αντιμετώπισης καλείται εφαρμογοκεντρικός (document-centered view) και αποτελεί ένα υψηλού επιπέδου σύστημα πολυδιεργασίας ή συνεπιτέλεσης (multitasking).

Επιπλέον μέσα από την εφαρμογή μας μπορούμε να έχουμε πρόσβαση και επικοινωνία με άλλου τύπου εφαρμογές για την ανάκτηση και ανάλυση δεδομένων διαφορετικής μορφής. Συγκεκριμένα, μέσα από μια εφαρμογή μας, μπορούμε να επεξεργαστούμε τα δεδομένα ενός λογιστικού φύλλου, να μορφοποιήσουμε και να εκτυπώσουμε τμήματα κειμένου που έχουν γραφτεί με έναν επεξεργαστή κειμένου, να χρησιμοποιήσουμε ένα πρόγραμμα επικοινωνίας για να στείλουμε ή να δεχτούμε μηνύματα, να αναλύσουμε τα δεδομένα μιας υπάρχουσας βάσης δεδομένων, να απεικονίσουμε γραφικά τα δεδομένα μας που προέρχονται από ένα πρόγραμμα παρουσιάσεων (σχήμα 11.11).

Οι σύγχρονες τεχνολογίες σύνδεσης, μας επιτρέπουν μέσα από τη δική μας εφαρμογή την αξιοποίηση των δεδομένων μιας άλλης ανεξάρτητης εξωτερικής εφαρμογής. Υπάρχουν δύο τρόποι επικοινωνίας εφαμοργών, φαινομενικά ίδιοι από την πλευρά του χρήστη αλλά με τεχνικές διαφοροποιήσεις.



Σχ. 11.11. Συνεργασία διαφορετικών τύπων εφαρμογών

Με τον πρώτο τρόπο, έχουμε τη δυνατότητα **διασύνδεσης** (linking) των δεδομένων μιας άλλης εφαρμογής με τη δική μας εφαρμογή. Τα δεδομένα της άλλης εφαρμογής που χρησιμοποιούμε στη δική μας, βρίσκονται σε κάποιο ξεχωριστό αρχείο του συστήματος. Με την τεχνική αυτή, η εφαρμογή μας καταφεύγει στο αρχείο και αντλεί τα δεδομένα, όποτε τα χρειαστεί, χωρίς να επιβαρύνεται σε χωρητικότητα. Τα δεδομένα αυτά της άλλης εφαρμογής παραμένουν προσπελάσιμα σε οποιαδήποτε τρίτη εφαρμογή μπορεί να τα χρειαστεί. Για παράδειγμα ένα αρχείο εικόνας είναι δυνατό να προσπελαύνεται από την εφαρμογή δημιουργίας του φυσικά, και ταυτόχρονα να είναι διασυνδεδεμένο τόσο στην εφαρμογή μας, όσο και σε κάποια τρίτη εφαρμογή. Κάθε χρονική στιγμή και στις τρεις εφαρμογές θα εμφανίζεται το πιο πρόσφατα ενημερωμένο αρχείο εικόνας.

Η τεχνική της διασύνδεσης, είναι ιδιαίτερα χρήσιμη για την υποστήριξη συνδέσεων πραγματικού χρόνου (real-time) μεταξύ εφαρμογών και για την κοινή προσπέλαση δεδομένων από διαφορετικές εφαρμογές με άμεση ενημέρωση των τροποποιήσεων του αντικειμένου. Μειονέκτημα αυτής της τεχνικής, είναι ότι πιθανή μετακίνηση των δεδομένων από το σημείο του συστήματος στο οποίο βρίσκονται, έχει σαν αποτέλεσμα την αδυναμία πρόσβασης της εφαρμογής μας στα δεδομένα αυτά, αν προηγουμένως δεν γίνει ενημέρωση του συνδέσμου: Χαρακτηριστικό του τρόπου αυτού σύνδεσης είναι ότι η τροποποίηση των δεδομένων της εξωτερικής εφαρμογής μπορεί

να γίνει μόνο μέσα από την εφαρμογή που τα δημιούργησε με ενεργοποίησή της είτε εξωτερικά, είτε εσωτερικά από την εφαρμογή μας

Ο δεύτερος τρόπος μας επιτρέπει την **ενσωμάτωση** (embedding) των δεδομένων μέσα στη δική μας εφαρμογή. Με την τεχνική αυτή εξασφαλίζεται ο αποκλειστικός έλεγχος των δεδομένων μέσα από την εφαρμογή μας. Τα δεδομένα δημιουργούνται μέσα από την εφαρμογή μας ή αντιγράφονται από κάποιο υπάρχον αρχείο και αποθηκεύονται στην εφαρμογή μας. Αυτό πρακτικά σημαίνει επιβάρυνση της εφαρμογής μας από την άποψη πως το αρχείο που δημιουργείται έχει μεγάλο μέγεθος, αλλά ταυτόχρονα εξασφαλίζεται η ασφάλεια των δεδομένων, αφού δεν επιτρέπεται να τροποποιηθούν από κάποια άλλη εφαρμογή.

Με την τεχνική της ενσωμάτωσης δύο εκδοχές είναι δυνατές. Τα δεδομένα να δημιουργηθούν εκείνη τη στιγμή ή να είναι ήδη έτοιμα. Στην πρώτη περίπτωση τα δεδομένα δημιουργούνται με τη χρήση μιας άλλης εφαρμογής που ενεργοποιείται μέσα από τη δική μας και ενσωματώνονται κατ' ευθείαν στην εφαρμογή μας. Για παράδειγμα, με την ενεργοποίηση μέσα από την εφαρμογή μας και τη χρήση ενός προγράμματος επεξεργασίας εικόνας, μπορούμε να δημιουργήσουμε ένα γραφικό που θα ενσωματωθεί στα δεδομένα της εφαρμογής μας. Σε αυτήν την περίπτωση δεν δημιουργείται κάποιο ξεχωριστό αρχείο στο σύστημα και επιπλέον δεν υπάρχει η δυνατότητα πρόσβασης από τη μεριά μιας τρίτης εφαρμογής σε αυτό το γραφικό.

Η βασική διαφορά μεταξύ των τεχνικών διασύνδεσης και ενσωμάτωσης είναι ο τρόπος αποθήκευσης των συνδεόμενων δεδομένων.

Στη δεύτερη περίπτωση, υπάρχει ανεξάρτητο αρχείο στο σύστημα, από το οποίο και προέρχονται τα δεδομένα που χρησιμοποιούμε στην εφαρμογή μας. Αυτό βέβαια δεν σημαίνει ότι το ανεξάρτητο αυτό αρχείο δεσμεύεται από την εφαρμογή μας. Ένα αντίγραφό του ενσωματώνεται στο αρχείο της εφαρμογής μας. Αν για παράδειγμα έχουμε διασυνδέσει την εφαρμογή μας με ένα λογιστικό φύλλο και κάποια στιγμή θέλουμε να επιφέρουμε κάποιες αλλαγές, ενεργοποιούμε μέσα από την εφαρμογή μας την εφαρμογή επεξεργασίας λογιστικών φύλλων και πραγματοποιούμε τις επιθυμητές αλλαγές. Αν στην συνέχεια "ανοίξουμε" το ανεξάρτητο αρχείο του λογιστικού φύλλου που υπάρχει στο σύστημα, θα διαπιστώσουμε ότι οι αλλαγές αυτές δεν έχουν καταγραφεί σε αυτό.

Ανακεφαλαιώνοντας, τα δεδομένα που συνδέονται με τη εφαρμογή μας με τη τεχνική της διασύνδεσης αποθηκεύονται στην εφαρμογή η οποία τα δημιούργησε, ενώ αντίθετα τα δεδομένα που συνδέονται στην εφαρμογή μας με τη τεχνική της ενσωμάτωσης, περιέχονται στην εφαρμογή μας και αποθηκεύονται μαζί της.

Ανακεφαλαίωση

Σε ένα σύγχρονο προγραμματιστικό περιβάλλον όπως είναι αυτά του αντικειμενοστραφούς και του οδηγούμενου από γεγονότα προγραμματισμού, προσφέρονται νέοι τρόποι υλοποίησης των εφαρμογών, νέα εργαλεία και τεχνικές, αυξημένο γραφικό περιβάλλον εργασίας, αλλά παράλληλα οι κλασικές τεχνικές του τμηματικού και του δομημένου προγραμματισμού είναι πάντοτε απαραίτητες και εφαρμόσιμες.

Μπορεί κάποιος να αναρωτηθεί, αν ένα σύγχρονο περιβάλλον κάνει δύσκολη τη ζωή των προγραμματιστών. Η απάντηση σίγουρα είναι αρνητική, αφού ο προγραμματιστής έχει πλέον ως εφόδιά του ένα σύνολο βοηθημάτων που τον επικουρούν, τον επιβλέπουν και τον διορθώνουν σε όλη τη διάρκεια της προγραμματιστικής εργασίας του. Φυσικά στην αρχή, ένας προγραμματιστής θα δυσκολευτεί με το τρόπο που πρέπει να συνδυάσει όλες τις νέες τεχνικές προγραμματισμού, αλλά πολύ σύντομα θα εξοικειωθεί και θα σκέφτεται κάθε εφαρμογή του με όρους αντικειμένων και γεγονότων.

Το μεγαλύτερο πλεονέκτημα ενός σύγχρονου προγραμματιστικού περιβάλλοντος, είναι ότι μας επιτρέπει να δημιουργήσουμε στις εφαρμογές μας ένα φιλικό, εύχρηστο και ευχάριστο περιβάλλον εργασίας για κάθε χρήστη. Τα γραφικά αντικείμενα που μας παρέχουν τα περιβάλλοντα αυτά συνδράμουν προς αυτήν την κατεύθυνση.

Σε ένα μοντέρνο και ολοκληρωμένο περιβάλλον, έχουμε τη δυνατότητα να επιτύχουμε την αρμονική συνεργασία διαφορετικών εφαρμογών. Μέσα από τη δική μας εφαρμογή μπορούμε να εμφανίσουμε και να επεξεργαστούμε δεδομένα από διαφορετικές πηγές. Επικοινωνώντας με τις υπόλοιπες εφαρμογές του συστήματος γλιτώνουμε χρόνο και κόστος, αλλά κυρίως εκμεταλλευόμαστε εφαρμογές που ήδη λειτουργούν σωστά και ο χρήστης είναι εξοικειωμένος με την χρήση τους.

Λέξεις κλειδιά

Αντικειμενοστραφής προγραμματισμός, οδηγούμενος από γεγονότα προγραμματισμός, αντικείμενα, κλάσεις, μέθοδοι, γεγονότα, τρόπος επικοινωνίας χρήστη-εφαρμογής, γραφικό περιβάλλον επικοινωνίας, επικοινωνία εφαρμογών, μενού επιλογών, πλαίσια διαλόγου, διασύνδεση, ενσωμάτωση.

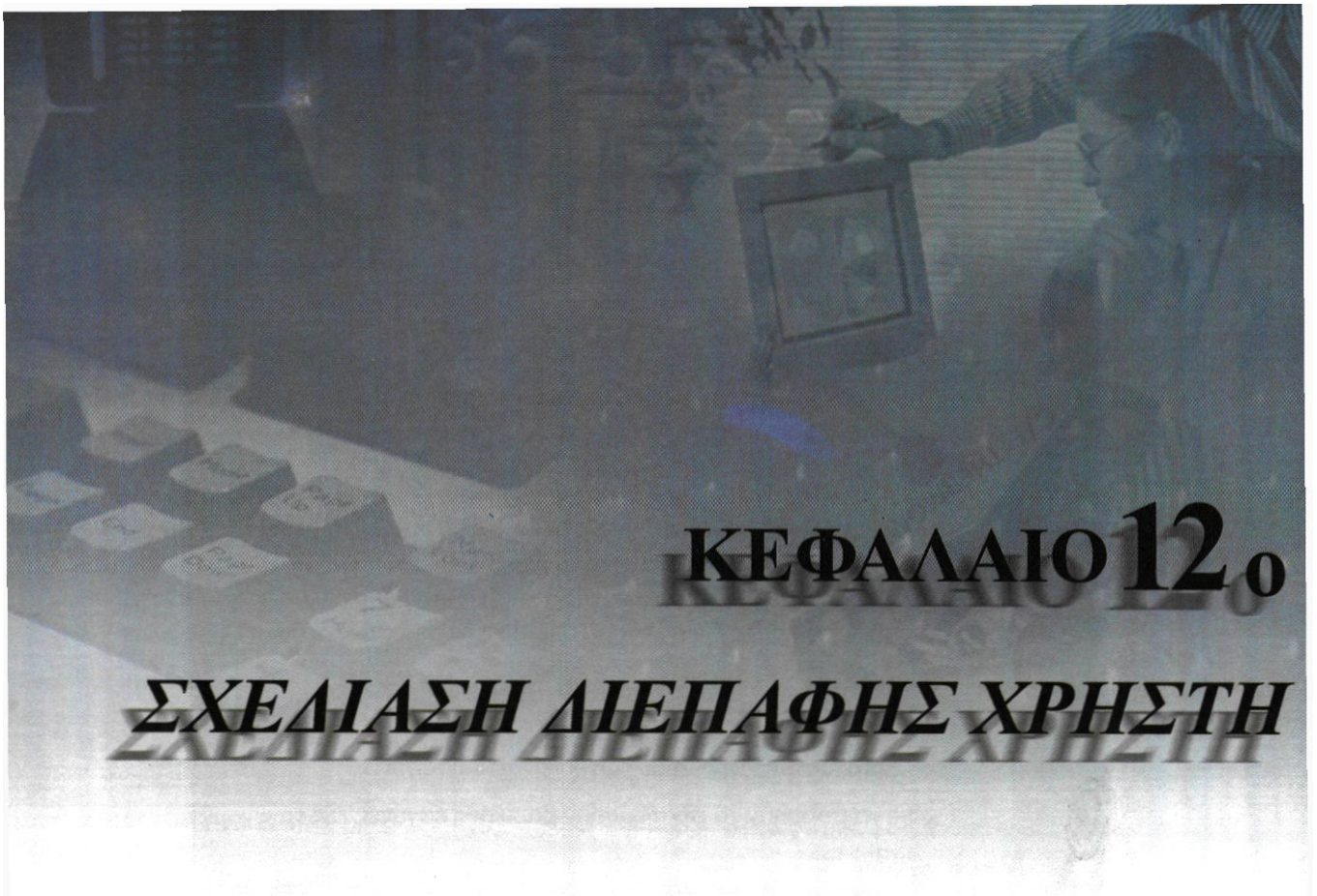
Ερωτήσεις - Θέματα για συζήτηση

1. Να γίνει συζήτηση σχετικά με τα πλεονεκτήματα που μας προσφέρουν τα σύγχρονα περιβάλλοντα προγραμματισμού.
2. Περιγράψτε τη μεθοδολογία ανάπτυξης μιας σύγχρονης εφαρμογής.
3. Περιγράψτε τη ροή εκτέλεσης μιας εφαρμογής που διέπεται από τις αρχές του αντικειμενοστραφή και του οδηγούμενου από γεγονότα προγραμματισμού.
4. Περιγράψτε γραφικά αντικείμενα που είναι δυνατό να χρησιμοποιήσετε στις εφαρμογές σας.
5. Δώστε τον ορισμό του αντικειμένου.
6. Δώστε παραδείγματα αντικειμένων σε προγραμματιστικό επίπεδο.
7. Τι είναι οι μέθοδοι;
8. Δώστε τον ορισμό του γεγονότος.
9. Από που αντλούν τα αντικείμενα τις αρχικές ιδιότητες και την συμπεριφορά τους;
10. Περιγράψτε παραδείγματα μενού επιλογών που είναι δυνατό να χρησιμοποιήσετε σε μια εφαρμογή;
11. Ποια είναι τα πλεονεκτήματα των προκαθορισμένων πλαισίων διαλόγου;
12. Ποια η διαφορά ενός ενσωματωμένου από ένα συνδεδεμένο αντικείμενο;
13. Να γίνουν εκτιμήσεις για τις μελλοντικές εξελίξεις στο χώρο του προγραμματισμού.

Βιβλιογραφία

1. Object-Oriented modeling and Design, Prentice Hall, New York, 1991.
2. G. Masini, A. Napoli, D. Colnet, D. Lionard, K. Tombre : Les langages á objets, InterEditions, Paris, 1989.
3. Gary Entsminger: Secrets of the Visual Basic for Windows, SAMS Publishing, Indiana USA, 1992.
4. Bertrand Meyer : Object-oriented Software Construction, Prentice Hall, London, 1988.

5. Παν. Πολίτης-Ηλ. Γιαννόπουλος: Προγραμματισμός με τη Visual Basic 4.0, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1997.



ΚΕΦΑΛΑΙΟ 12ο

ΣΧΕΔΙΑΣΗ ΔΙΕΠΑΦΗΣ ΧΡΗΣΤΗ

ΠΕΡΙΕΧΟΜΕΝΑ

- Διεπαφή χρήστη
- Τύποι διεπαφής χρήστη
- Γενική σχεδίαση διεπαφής χρήστη
- Οπτική σχεδίαση διεπαφής χρήστη
- Ηχητική σχεδίαση διεπαφής χρήστη

Εισαγωγή

Το θέμα της σχεδίασης του τρόπου με τον οποίο ένα τμήμα λογισμικού έρχεται σε επαφή με το χρήστη είναι πρωταρχικής σημασίας. Οι υπολογιστικές ή οποιουδήποτε άλλου είδους ικανότητες που έχει το σύστημα, καθώς και οι χρηστικές δυνατότητες που παρέχει, μπορεί πολύ εύκολα να υποτιμηθούν και να μείνουν αναξιοποίητες αν η διεπαφή χρήστη (user interface) της εφαρμογής δεν είναι προσεκτικά σχεδιασμένη και υλοποιημένη. Οι αρχές εργονομίας, υλικού και λογισμικού, καθορίζουν εκείνες τις προδιαγραφές που θα πρέπει να ακολουθούνται ώστε να επιτυγχάνεται από την μια μεριά ελαχιστοποίηση της καταπόνησης του χρήστη και από την άλλη μεγιστοποίηση της ωφελιμότητας από την χρήση του συστήματος. Θα πρέπει να γίνει κατανοητό πως ο τρόπος επικοινωνίας του συστήματος με το χρήστη είναι εξ ίσου σημαντικός με το ίδιο το περιεχόμενό του.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές :

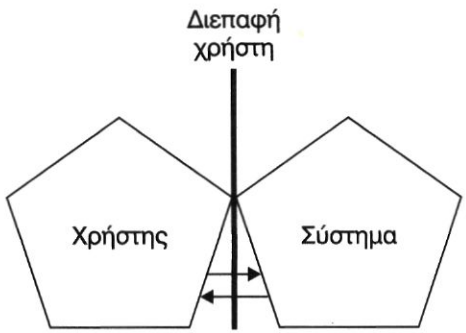
- να μπορούν να εφαρμόσουν βασικούς κανόνες της εργονομίας λογισμικού
- να αναπτύξουν δεξιότητες σχεδιασμού περιβάλλοντος διεπαφής
- να μπορούν να υλοποιούν ένα απλό, φιλικό και εύχρηστο περιβάλλον διεπαφής

Προερωτήσεις

- ✓ θεωρείς αποτρεπτικό στοιχείο για να ασχοληθείς με ένα πρόγραμμα το γεγονός ότι παρουσιάζει ανομοιομορφία σε σχέση με ότι μέχρι σήμερα έχεις συνηθίσει;
- ✓ συμμερίζεσαι την άποψη ότι ένα λογισμικό είναι τόσο πιο χρήσιμο, όσο πιο απλό είναι στη χρήση του;
- ✓ νομίζεις ότι υπάρχουν κανόνες σχεδίασης λογισμικού;
- ✓ πιστεύεις ότι η επιλογή χρωμάτων σε ένα λογισμικό στηρίζεται αποκλειστικά και μόνο στο προσωπικό γούστο του καθένα;
- ✓ όταν κάνεις λάθος χειρισμό σε ένα πρόγραμμα, τι θα ήθελες να κάνει στη συνέχεια το πρόγραμμα για να σε βοηθήσει;

12.1. Διεπαφή χρήστη

Ο όρος διεπαφή χρήστη (user interface) είναι το σύνολο των συστατικών ενός συστήματος το οποίο επιτρέπει αμφίδρομη επικοινωνία μεταξύ συστήματος και χρήστη. Η διεπαφή χρήστη ενός συστήματος έχει σχέση με το ίδιο το σύστημα, το χρήστη του συστήματος και τον τρόπο που αλληλεπιδρούν μεταξύ τους (σχήμα 12.1). Ο όρος θέλει να δείξει το σημείο επαφής χρήστη και υπολογιστή, την γραμμή επαφής πίσω από την μια μεριά της οποίας βρίσκεται η μηχανή και πίσω από την άλλη μεριά ο άνθρωπος. Έτσι λοιπόν η διεπαφή χρήστη περιέχει στοιχεία που είναι τμήματα τόσο του υλικού του συστήματος, όσο και του λογισμικού που "τρέχει" σε αυτό.



Σχ. 12.1 : Σχηματική αναπαράσταση διεπαφής χρήστη

Ως στοιχεία του υλικού του συστήματος που περιλαμβάνονται στη διεπαφή χρήστη μπορούν να αναφερθούν μια οθόνη επαφής, μια φωτογραφίδα (lightpen) ή ένα ποντίκι. Μέρη του λογισμικού της διεπαφής χρήστη είναι, για παράδειγμα, τα μηνύματα λάθους, τα ηχητικά μηνύματα, τα εργαλεία πλοήγησης, εικόνες σύμβολα και αντικείμενα πάνω στην οθόνη, κάθε τι που διαθέτει το λογισμικό σαν στοιχείο αλληλεπίδρασης του συστήματος με το χρήστη. Με άλλα λόγια, ο όρος σε ότι αφορά τη λογισμική του υπόσταση, σημαίνει ένα σύνολο από οπτικές και ακουστικές παραμέτρους, που παρέχει ο υπολογιστής προς το χρήστη, μέσω του εκάστοτε εκτελούμενου προγράμματος, με σκοπό την καλύτερη επικοινωνία και συνεργασία μεταξύ ανθρώπου και μηχανής. Μπορούμε να παρομοιάσουμε τη διεπαφή χρήστη σαν κανάλι επικοινωνίας μεταξύ χρήστη και υπολογιστή.

Στο παρόν κεφάλαιο η εστίαση του ενδιαφέροντός μας γίνεται μόνο στα στοιχεία εκείνα που αναφέρονται στο λογισμικό.

Κάθε χρονική στιγμή στην καθημερινή μας ζωή, μας περιβάλλουν διάφορα προϊόντα με καλά ή άσχημα σχεδιασμένη διεπαφή χρήστη. Όσο τα

Στο εγκυρότατο λεξικό Ηλεκτρικών όρων, IEEE Standard Directory of Electrical and Electronics Terms, η έννοια της λέξης **interface** αποδίδεται με τον όρο **shared boundary,** ο οποίος μεταφράζεται "κοινό ή διαμοιραζόμενο όριο ή σύνορο".

Σε έκδοση της ΕΠΥ (Ελληνική Εταιρεία Επιστημόνων Ηλεκτρονικών Υπολογιστών και Πληροφορικής) ο όρος interface αποδίδεται με τη λέξη διασύνδεση. Ακόμα για την απόδοση του όρου interface χρησιμοποιούνται οι λέξεις επικοινωνία, τρόπος επικοινωνίας, αντιμετώπιση και προσαρμογή. Σε πιο πρόσφατες ερμηνείες ο όρος στα ελληνικά αποδίδεται ως διεπαφή χρήστη, όρος ο οποίος είναι ο επικρατέστερος.

προϊόντα γίνονται πιο σύνθετα, ειδικά όταν πρόκειται για ηλεκτρικές και ηλεκτρονικές συσκευές, τόσο πιο συχνά ο χρήστης απογοητεύεται από τη δυσκολία που παρουσιάζει ο τρόπος χρήσης τους και πολύ συχνά δεν έχει τη διάθεση να κάνει καμία προσπάθεια για να τα κατανοήσει και να τα χειριστεί.

Μέσα στον όρο επικοινωνία με το χρήστη περιλαμβάνεται η έννοια της αλληλεπίδρασης του χρήστη με το προϊόν. Τι μπορεί να κάνει ο χρήστης με το προϊόν, και τι μπορεί να κάνει το προϊόν για το χρήστη, είναι μέρη της επικοινωνίας του χρήστη με το προϊόν. Η χρησιμότητα ή μη ενός προϊόντος εξαρτάται πολλές φορές από την ικανότητα του καταναλωτή να το χειριστεί με επιτυχία. Εάν ένας χρήστης δεν μπορεί να εξακριβώσει πως λειτουργεί ένα προϊόν, τότε η επικοινωνία του προϊόντος με το χρήστη είναι ελλειπής και η χρηστικότητα του προϊόντος θεωρείται αυτόματα αποτυχημένη.

12.2. Τύποι διεπαφής χρήστη

Η μεγάλη ποικιλία περιβαλλόντων λογισμικού, έφερε σαν επακόλουθο την παραγωγή και παρουσία, ιδιαίτερα μέχρι την προηγούμενη δεκαετία, διαφορετικών τύπων διεπαφής χρήστη. Το πρόβλημα που προέκυψε αφορούσε κύρια το υλικό, αφού ο φυσικός έλεγχός του (πληκτρολόγιο, ποντίκι κλπ) είναι λογικός και συγκεκριμένος, αλλά τα προβλήματα άρχιζαν από τις πολλές ποικιλίες λογισμικού, που είχαν σαν αποτέλεσμα τη συχνότατη εναλλαγή παραστάσεων στην οθόνη του υπολογιστή.

Ο τύπος διεπαφής χρήστη που αρχικά παρουσιάστηκε ήταν **βασισμένος σε εντολή** (command-based). Ο χρήστης για να επικοινωνήσει με τον υ-

πολογιστή, έπρεπε να πληκτρολογήσει κάποια εντολή ή μια σειρά από εντολές, ανάλογα με το περιβάλλον λογισμικού στο οποίο βρισκόταν. Χαρακτηριστικό παράδειγμα τέτοιου τύπου διεπαφής χρήστη παρείχε το λειτουργικό σύστημα DOS. Αυτό το είδος διεπαφής χρήστη απαιτούσε περισσότερα από τον ανθρώπινο παράγοντα, παρά από τον υπολογιστή. Ο χρήστης έπρεπε να είναι ειδικός στη γλώσσα που υποστήριζε το περιβάλλον εργασίας, ώστε να μπορεί να μεγιστοποιηθεί η απόδοση του υπολογιστή. Μερικοί κατασκευαστές υπολογιστών προσπάθησαν να ξεπεράσουν αυτό το πρόβλημα θέτοντας κάποιες άλλες προδιαγραφές στη σχεδίαση του τρόπου επικοινωνίας χρήστη και υπολογιστή.

Μία προσπάθεια έγινε και στο ίδιο το περιβάλλον του λειτουργικού συστήματος DOS, με την εμφάνιση του κελύφους DOS (dosshell), αλλά και άλλων προγραμμάτων που λειτουργούν στο περιβάλλον DOS (Norton Utilities, κλπ). Αυτά τα προγράμματα λογισμικού, αν και η φιλοσοφία λειτουργίας τους παραμένει η ίδια, εντούτοις εμφανίζουν μια περισσότερο φιλική και εύχρηστη διεπαφή χρήστη, προσδίδοντας μια άλλη όψη στο πολύ αυστηρό περιβάλλον.

Η πιο επιτυχής όμως από όλες τις προσπάθειες ήταν αυτή της Apple. Η Apple ήταν η πρώτη που εισήγαγε τη χρήση εικονιδίων και γραφημάτων στις διαδικασίες επικοινωνίας χρήστη και υπολογιστή, δημιουργώντας την έννοια της **γραφικής** διεπαφής χρήστη **GUI** (Graphical User Interface). Αυτή η μορφή διεπαφής χρήστη αυξάνει τη λειτουργικότητα, την ευκολία στη χρήση, την αποτελεσματικότητα και την ταχύτητα του διαλόγου μεταξύ ανθρώπου και μηχανής.

Στην περίπτωση των προσωπικών υπολογιστών IBM και συμβατών, τα PCs, καταρχήν δεν υπήρξαν σταθερές ως προς την σχεδίαση της διεπαφής χρήστη. Οι σταθερές αυτές αναπτύχθηκαν αργότερα για λειτουργικά συστήματα που δουλεύουν χρησιμοποιώντας γραφικές μορφές επικοινωνίας με το χρήστη, όπως είναι τα Windows και το OS/2. Και τα δύο αυτά προϊόντα έχουν ως πρότυπο το περιβάλλον της Apple. Γεγονός πάντως είναι πως οι υπολογιστές Macintosh της Apple αναγνωριζόντουσαν μέχρι πριν από λίγα χρόνια απ' όλους σαν οι περισσότερο συνεπείς προσωπικοί υπολογιστές στο θέμα της διεπαφής χρήστη.

Τα GUIs θεωρούν ότι ο κόσμος αποτελείται από αντικείμενα και ότι ενέργειες εφαρμόζονται πάνω τους. Ακόμα, σε επίπεδο έκφρασης της επιλογής του χρήστη, τα clicks του ποντικιού αντικαθιστούν τις πληκτρολογήσεις.

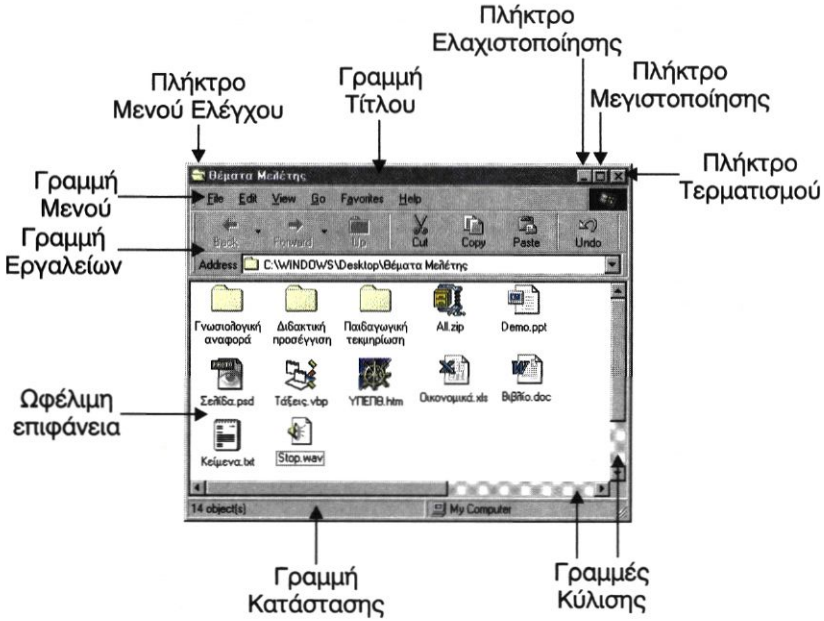
Το GUI είναι σήμερα ο πιο διαδεδομένος και δημοφιλής τύπος διεπαφής χρήστη. Η θεώρηση που επιτελεί, και που αναφέρεται παραπάνω, συμ-

φωνεί απόλυτα με τις θεωρήσεις του αντικειμενοστραφούς και του οδηγούμενου από γεγονός προγραμματισμού σχετικά με τα αντικείμενα και τα γεγονότα.

Χαρακτηριστικά γραφικών διεπαφών χρήστη

Τα βασικά χαρακτηριστικά των γραφικών διεπαφών χρήστη είναι:

- *Η επιφάνεια εργασίας*: Είναι η γραφική διαμόρφωση της οθόνη του υπολογιστή, μετατρέποντάς την σε ένα "ηλεκτρονικό γραφείο".
- *Τα παράθυρα* : Ένα παράθυρο αποτελεί μια παραλληλόγραμμη ενιαία περιοχή δραστηριοτήτων στην οθόνη του υπολογιστή. Μέσω των παραθύρων πραγματοποιούνται οι κύριες εργασίες ανάκτησης και επεξεργασίας των πληροφοριών. Βασικά στοιχεία παραθύρων είναι (σχήμα 12.2):
  - ✓ η ωφέλιμη επιφάνεια, δηλαδή ο χώρος του παραθύρου μέσα στον οποίο μπορούν να ανακτώνται και να επεξεργάζονται διαφόρων τύπων δεδομένα,
  - ✓ η γραμμή τίτλου (title bar), στην οποία εμφανίζεται το όνομα του παραθύρου,



***Σχ. 12.2:** Βασικά στοιχεία παραθύρων.*

- ✓ η γραμμή μενού (menu bar), στην οποία εμφανίζονται λεκτικά οι ομαδοποιήσεις των δραστηριοτήτων που παρέχει το συγκεκριμένο παράθυρο,
- ✓ η γραμμή εργαλείων (toolbar), στην οποία παρουσιάζονται με εικονίδια οι πιο συχνά χρησιμοποιούμενες λειτουργίες που μπορεί να επιτελεί ο χρήστης μέσω του παραθύρου,
- ✓ η γραμμή κατάστασης (status bar), όπου εμφανίζονται πληροφορίες σχετικά με το παράθυρο και τα στοιχεία του,
- ✓ οι γραμμές κύλισης (scroll bars), οι οποίες είναι η οριζόντια και η κατακόρυφη, και επιτρέπουν την μετακίνηση του οπτικού πεδίου του χρήστη, στην περίπτωση που το εύρος του παραθύρου είναι μικρότερο της επιφάνειάς του,
- ✓ το πλήκτρο μενού ελέγχου (control menu), το οποίο περιλαμβάνει μενού επιλογών με εντολές χειρισμού του παραθύρου,
- ✓ το πλήκτρο μεγιστοποίησης (maximize button), που μεγιστοποιεί το μέγεθος του παραθύρου ώστε να καταλαμβάνει όλο το εύρος της οθόνης,
- ✓ το πλήκτρο ελαχιστοποίησης (minimize button), που έχει σαν αποτέλεσμα την μετατροπή του παραθύρου σε εικονίδιο,
- ✓ το πλήκτρο επαναφοράς (restore button), που επαναφέρει το παράθυρο στο προηγούμενό του μέγεθος,
- ✓ το πλήκτρο τερματισμού (close button), που κλείνει το παράθυρο.

- *Τα γραφικά αντικείμενα*: Αποτελούν τη βάση της γραφικής διεπαφής χρήστη. Κάθε αντικείμενο έχει ορισμένες ιδιότητες και με κάθε αντικείμενο ο χρήστης μπορεί να επιτελέσει ορισμένες εργασίες. Ένα αντικείμενο μπορεί να περιέχει άλλα αντικείμενα, τα οποία συνήθως ομαδοποιούνται με βάση κάποιο κριτήριο (λογικό, λειτουργικό, κλπ).
- *Οι δείκτες του ποντικιού*: Είναι μικρά γραφικά αντικείμενα που δείχνουν το σημείο στο οποίο πρόκειται να γίνει η επόμενη επαφή του χρήστη με το σύστημα. Οι μορφές του δείκτη είναι ποικίλες και αντιστοιχείται από μια σε κάθε διαφορετικού τύπου εργασία που εκτελεί το σύστημα.

12.3. Γενική σχεδίαση διεπαφής χρήστη

Η σχεδίαση ενός καλού τρόπου επικοινωνίας με τον χρήστη, είναι το μισό μιας καλής εφαρμογής, ειδικά στο χώρο των εφαρμογών πολυμέσων.

Προϋπόθεση για τη σχεδίαση μιας επιτυχημένης διεπαφής είναι το να έχει γίνει επακριβώς κατανοητός ο στόχος που πρόκειται να εξυπηρετήσει το λογισμικό και οι απαιτήσεις χρηστικότητας του χρήστη από αυτό. Ο καλύτερος τρόπος για να απαντηθεί η ερώτηση "τι θέλει ο χρήστης να κάνει με αυτή την εφαρμογή;", είναι να μελετηθεί στην πράξη η εργασία του χρήστη. Η διαδικασία ανάπτυξης πολλαπλών σεναρίων, διαφορετικών μεταξύ τους, που περιγράφει το καθένα ξεχωριστούς τρόπους αλληλεπίδρασης και επικοινωνίας με το χρήστη για την υλοποίηση της εφαρμογής και στη συνέχεια η εξαγωγή συμπεράσματος για ποιο από αυτά θα προωθήσει καλύτερα τους σκοπούς της εργασίας, πάντοτε από την πλευρά του χρήστη, είναι μια διαδικασία η οποία θα πρέπει να γίνει πριν από την έναρξη σχεδίασης της διεπαφής χρήστη και η οποία ονομάζεται ανάλυση εργασιών χρήστη (user task analysis).

Η επικοινωνία μιας εφαρμογής με το χρήστη είναι τόσο σημαντική, όσο και το ίδιο το περιεχόμενο της εφαρμογής. Χωρίς μια καλή διεπαφή χρήστη, πιθανά ο χρήστης δεν θα μπορέσει ποτέ να πάρει το πλήρες περιεχόμενο της εφαρμογής. Βασικές αρχές *Εργονομίας Λογισμικού* που θα πρέπει να τηρούνται κατά τη σχεδίαση μιας λειτουργικής διεπαφής χρήστη είναι οι παρακάτω:

Απλούστερο για το χρήστη, σημαίνει δυσκολότερο για το σχεδιαστή, αλλά αυτή η προσπάθεια αξίζει τον κόπο και απαιτείται να γίνει.

- *Συνέπεια*: Αν η ίδια διαδικασία, η ίδια ενέργεια, συντελείται σε δύο ή περισσότερα διαφορετικά μέρη της εφαρμογής, θα πρέπει να παρουσιάζεται και να λειτουργεί ακριβώς με τον ίδιο τρόπο σε όλα τα μέρη. Από τη στιγμή που ο χρήστης έχει διδαχθεί να επιτελεί μια εργασία, θα πρέπει κάθε φορά που καλείται να την επανεκτελέσει, να χρησιμοποιεί τον ίδιο τρόπο.
- *Απλότητα*: Αν υπάρχουν περισσότεροι από ένας τρόποι για να παρουσιαστεί μία διεργασία, θα πρέπει να επιλεγεί η απλούστερη. Αν υπάρχουν σύνθετες διεργασίες στην εφαρμογή, θα πρέπει να γίνει κάθε δυνατή προσπάθεια, έτσι ώστε να βρεθούν εκείνοι οι τρόποι παρουσίασης που θα τις κάνουν να φαίνονται και να λειτουργούν όσο το δυνατό απλούστερα.
- *Χρήση μεταφορών*: Πρέπει να χρησιμοποιούμαι προσεγγίσεις που είναι ήδη οικείες και γνωστές στο χρήστη. Για παράδειγμα, εάν πρόκειται να δημιουργηθεί μια εφαρμογή που να περιλαμβάνει video και πρόκειται να δοθεί στο χρήστη η δυνατότητα ελέγχου επάνω στη ροή του, τότε είναι λογικό και προτιμητέο να χρησιμοποιηθεί γραφική αναπαράσταση των πλήκτρων ελέγχου παρόμοια με εκείνα που χρησιμοποιούνται

στις οικιακές συσκευές video, οι οποίες είναι ήδη γνωστές και οικείες στο χρήστη.

- *Ελαχιστοποίηση ενεργειών χρήστη:* Ο χρήστης πρέπει να φτάνει στο επιθυμητό για αυτόν αποτέλεσμα με τις λιγότερες δυνατές ενέργειες. Οι απαιτούμενες πληκτρολογήσεις πρέπει να περιορίζονται στις απολύτως απαραίτητες. Στα σημεία εκείνα του προγράμματος που η επιλογή του χρήστη μπορεί να εκφραστεί με περισσότερους από έναν τρόπους θα πρέπει να επιλέγεται ο λιγότερο κοπιαστικός για το χρήστη. Για παράδειγμα, όπου στη ροή ενός προγράμματος χρησιμοποιούνται προεπιλεγμένες επιλογές, θα πρέπει να παρουσιάζονται με μορφή λίστας ή πλήκτρων επιλογής, έτσι ώστε ο χρήστης να μην υποχρεωθεί να πληκτρολογήσει την επιλογή του.

Αλλά και στις περιπτώσεις που η πληκτρολόγηση είναι απαραίτητη, οπότε φυσιολογικό είναι να γίνονται λάθη στην πληκτρολόγηση από τους χρήστες, το πρόγραμμα θα πρέπει επίσης να λειτουργεί ανάλογα. Για παράδειγμα, αν ο χρήστης πληκτρολογήσει και καταχωρήσει μια απάντηση - φράση έχοντας κάνει λάθος σε ένα σημείο της, το πρόγραμμα θα πρέπει να του επιτρέπει να διορθώσει το συγκεκριμένο σημείο και να μην τον αναγκάζει να επαναπληκτρολογήσει ολόκληρη την απάντηση.

- *Παροχή άμεσης ανάδρασης*: Ακόμα και στην περίπτωση που τα αποτελέσματα της επιλογής του χρήστη δεν έχουν ολοκληρωθεί, το σύστημα πρέπει να παρέχει κάποια μηνύματα στο χρήστη. Μερικές φορές ο χρήστης επιλέγει μια τέτοια διαδικασία που η ολοκλήρωση της απαιτεί κάποιο χρόνο. Εξαιτίας αυτής της καθυστέρησης είναι πολύ σημαντικό να δοθεί η δυνατότητα στον χρήστη να πάρει κάποιο άμεσο μήνυμα αποδοχής της επιλογής του με ταυτόχρονο αναγγελία ότι το αποτέλεσμα της ετοιμάζεται. Ένα μήνυμα στην οθόνη του τύπου "Παρακαλώ περιμένετε, κάνω υπολογισμούς", προσφέρει επαρκή πληροφόρηση και καθησυχάζει το χρήστη ότι όλα πάνε καλά.

Χρόνος απόκρισης του συστήματος ονομάζεται ο χρόνος που χρειάζεται το σύστημα να αρχίσει να παρέχει τα αποτελέσματα μιας ενέργειας ή μιας διαδικασίας που ζήτησε ο χρήστης

- *Παροχή βοήθειας*: Σημαντικό στοιχείο στην προσπάθεια χρήσης ενός προγράμματος είναι η επίγνωση από τη μεριά του χρήστη του τι μπορεί να κάνει κάθε στιγμή με το πρόγραμμα. Η βοήθεια που μπορεί να λαμβάνει ο χρήστης κατά τη διάρκεια χρήσης ενός προγράμματος μπορεί να είναι:

- ✓ on line βοήθεια, με την μορφή και τη λειτουργικότητα που παρέχεται από όλα τα σύγχρονα πακέτα λογισμικού. Ο χρήστης έχει τη δυνατότητα να ανατρέξει σε συνοπτικές οδηγίες χρήσης του προγράμματος, σε επιμέρους λειτουργίες, σε αναζήτηση, ανεύρεση και επεξήγηση όρων.
- ✓ άμεση βοήθεια, με τη μορφή των βοηθών (assistants) που παρέχουν οι τελευταίες εκδόσεις πολλών πακέτων λογισμικού. Ο χρήστης μπορεί να ζητήσει και να πάρει βασικές πληροφορίες σχετικά με τις δυνατότητες που παρέχει ο χώρος στον οποίο βρίσκεται και τις πρώτες ενέργειες που θα πρέπει να κάνει.
- ✓ έμμεση βοήθεια, μέσω σωστά και επεξηγηματικά διατυπωμένων μηνυμάτων λαθών.

- *Ελαχιστοποίηση απομνημόνευσης*: Ο χρήστης για να μπορέσει να αλληλεπιδράσει με το λογισμικό, δεν θα πρέπει να θυμάται παρά μόνο τα απολύτως απαραίτητα Η ποσότητα αυτή πρέπει να είναι η ελάχιστη δυνατή. Οι εργασίες πρέπει να είναι με τέτοιο τρόπο δομημένες, έτσι ώστε η ολοκλήρωσή τους να γίνεται μειώνοντας στο ελάχιστο την περίπτωση να ξεχάσει ο χρήστης κάποιο βήμα.

- *Εναρμόνιση*: Θα πρέπει να λαμβάνεται σοβαρά υπόψη η προηγούμενη εμπειρία και οι αναπαραστάσεις του χρήστη από άλλα προγράμματα λογισμικού. Έχοντας κατά νου πως όλο και περισσότερο έντονα εμφανίζεται μία τάση τυποποίησης ενεργειών και διαδικασιών, θα πρέπει τα προγράμματά μας να εναρμονίζονται με αυτές. Για παράδειγμα, αν το πρόγραμμά μας, με το πάτημα ενός γραφικού πλήκτρου, εκτελεί διαδικασία αποθήκευσης, το εικονίδιο που τυχόν θα έχουμε αποδώσει στο πλήκτρο αυτό θα πρέπει να είναι μια δισκέτα. Σχεδόν όλα τα πακέτα λογισμικού χρησιμοποιούν αυτό το εικονίδιο για τη συγκεκριμένη διαδικασία και για το χρήστη θα είναι πολύ εύκολο βλέποντάς το να έχει τη σωστή αναπαράσταση της λειτουργίας που επιτελεί.

Ο σχεδιαστής της εφαρμογής πρέπει να ακολουθεί τις συνήθειες του χρήστη και όχι ο χρήστης να ακολουθεί τις συνήθειες του σχεδιαστή.

- *Ευκαμψία:* Το πρόγραμμα θα πρέπει να παρουσιάζει ευκαμψία στις ενέργειες και στις πληκτρολογήσεις του χρήστη. Για παράδειγμα, σε ένα πρόγραμμα που ο χρήστης καλείται να απαντήσει σε ερωτήσεις του τύπου "Θέλεις να γίνει η καταχώρηση; (Ν/Ο)", "Να γίνει η διαγραφή; (Ν/Ο)", η σωστή νοηματικά απάντησή του θα πρέπει να γίνεται αποδεκτή ανεξάρτητα από την κατάσταση στην οποία βρίσκεται το πληκτρολόγιο (ελληνικά ή αγγλικά, κεφαλαία ή πεζά).

Το παράδειγμα αυτό παρουσιάζεται αμέσως στη συνέχεια σε επίπεδο **ΓΛΩΣΣΑΣ**.

Παράδειγμα

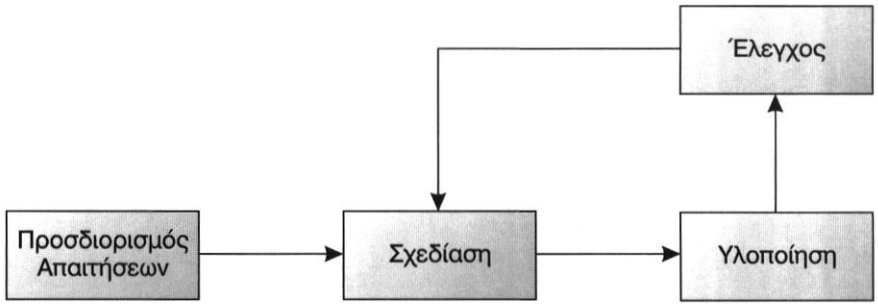
```
.................
! Εισαγωγή στοιχείων
ΔΙΑΒΑΣΕ ΟΝΟΜΑ, ΕΠΩΝΥΜΟ, ΔΙΕΥΘΥΝΣΗ, ΤΗΛΕΦΩΝΟ
ΚΑΛΕΣΕ Επικύρωση (Σ)
.................

ΔΙΑΔΙΚΑΣΙΑ Επικύρωση(σ)
σ <- 0
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ "Να γίνει καταχώρηση των στοιχείων (Ν/Ο);"
    ΔΙΑΒΑΣΕ Κ
    ΕΠΙΛΕΞΕ Κ
        ΠΕΡΙΠΤΩΣΗ "Ν" , "ν", "Ν", "n"
            σ <- 1
        ΠΕΡΙΠΤΩΣΗ "Ο", "ο", "Ο", "ο"
            σ <- 2
        ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
            ΓΡΑΨΕ "Πιέστε το Ν ή το Ο"
    ΤΕΛΟΣ ΕΠΙΛΟΓΩΝ
ΜΕΧΡΙΣ_ΟΤΟΥ σ=1 ΄Η σ=2
ΤΕΛΟΣ Επικύρωση
```

Στη λύση θεωρήσαμε ότι είναι σημαντικό ο χρήστης να απαντήσει αποκλειστικά με ένα ναι ή ένα όχι, γι' αυτό και ο έλεγχος γίνεται αναλυτικά και για τα δύο ενδεχόμενα.

Η ενσωμάτωση των παραπάνω χαρακτηριστικών στα προγράμματα λογισμικού, έχει σαν αποτέλεσμα να καθιστά το πρόγραμμα ιδιαίτερα φιλικό προς το χρήστη, με συνέπεια την αύξηση της διάθεσης του χρήστη για ενασχόλησή του με το πρόγραμμα. Ένα φιλικό πρόγραμμα γίνεται γρήγορα αποδεκτό. Η κατασκευή φιλικών προγραμμάτων, απαιτεί από την πλευρά του σχεδιαστή την επιστράτευση γνώσεων και φαντασίας, ώστε να μπορέσει να επιτύχει το καλύτερο δυνατό αποτέλεσμα για λογαριασμό του χρήστη.

Φάσεις ανάπτυξης της διεπαφής χρήστη

Η ανάπτυξη της διεπαφής χρήστη γίνεται με τέτοιο τρόπο ώστε το κέντρο ενδιαφέροντος και προβληματισμού να είναι ο χρήστης. Μια βασική μεθοδολογία για την ανάπτυξη της διεπαφής χρήστη είναι η δημιουργία πρωτοτύπου. Με τον όρο *πρωτότυπο* εννοούμε ένα απλουστευμένο μοντέλο του συστήματος που έχει επινοηθεί με σκοπό να περιγράψει τη λειτουργικότητά του. Η μεθοδολογία δημιουργίας πρωτοτύπου περιλαμβάνει τρεις φάσεις (σχήμα 12.3) :



Σχ. 12.3. Φάσεις ανάπτυξης διεπαφής χρήστη

- *Προσδιορισμός απαιτήσεων*, όπου μελετάται και αναλύεται το θέμα, προσδιορίζεται ο σκοπός του λογισμικού και οι στόχοι που θέτει, και αναγνωρίζεται και περιγράφεται τόσο το περιβάλλον στο οποίο θα λειτουργήσει όσο και οι χρήστες που θα εξυπηρετήσει.
- *Σχεδίαση*, όπου επιλέγεται η επικρατέστερη μορφοποίηση διεπαφής χρήστη μεταξύ των προτεινομένων. Η επιλογή λαμβάνει υπόψη της το κόστος, το χρόνο εκμάθησης, τις διαθέσιμες πηγές υλικού. Δημιουργούνται σαφώς τα αντικείμενα που θα το υλοποιούν, οι ενέργειες που θα επιτελεί κάθε ένα από αυτά και τα γεγονότα στα οποία θα αντιδρά.
- *Υλοποίηση*, όπου τα όσα περιγράφονται στη σχεδίαση υλοποιούνται με τη χρήση των κατάλληλων εργαλείων. Δημιουργούνται τα απαραίτητα γραφικά, τα μηνύματα λάθους, τα ηχητικά δεδομένα.
- *Έλεγχος*, όπου πειραματικά ελέγχεται η αποτελεσματικότητα της διεπαφής χρήστη σε πιλοτική χρήση με ακροατήριο ανάλογο αυτού που πραγματικά απευθύνεται και ξεκινά μία διαδικασία βελτίωσης στα σημεία που κρίνεται απαραίτητο, με επάνοδο στη φάση της σχεδίασης για την πραγματοποίηση των απαιτούμενων βελτιώσεων.

12.4. Οπτική σχεδίαση της διεπαφής χρήστη

Από τα βασικότερα στοιχεία τα οποία πρέπει με προσοχή να συνεκτιμώνται και να λαμβάνονται υπόψη κατά τη δημιουργία της διεπαφής χρήση μιας εφαρμογής, είναι όσα σχετίζονται με την επιλογή χρωμάτων των στοιχείων της εφαρμογής και τη σχεδίαση των μηνυμάτων λαθών.

12.4.1. Το χρώμα

Το χρώμα θεωρείται βασικό συστατικό στη σχεδίαση της οθόνης μιας εφαρμογής. Η σωστή χρήση του χρώματος μπορεί να αποδειχθεί ένας αποτελεσματικός μηχανισμός για την επικοινωνία, την προσοχή και τον καθορισμό των σημείων που θέλει να αναδείξει μία εφαρμογή. Η χρήση του χρώματος θα πρέπει να γίνεται με προσοχή και με βάση την κοινή ανθρώπινη αίσθηση αντίληψης των διαφορών μεταξύ των χρωμάτων, των πληροφοριών που υποσυνείδητα λαμβάνονται από αυτά και των βασικών αρχών που ορίζει η ψυχολογία χρωμάτων. Το χρώμα εκλύει τους χρήστες, αλλά από τη λανθασμένη χρήση του μπορεί να προκληθούν αρνητικά αποτελέσματα.

Το χρώμα έχει τρεις βασικούς άξονες χρήσης : αναγνώριση, αντίθεση, επικέντρωση. Η εμπειρία όλων μας λέει πως το χρώμα μπορεί να γοητεύει, αλλά μπορεί και να απωθεί. Τα φωτεινά, ζωηρά χρώματα, τα χρώματα που κάνουν αντίθεση με οτιδήποτε άλλο, τραβούν την προσοχή. Οι λειτουργίες που επιτελεί το χρώμα είναι :

- να προκαλεί υποσυνείδητα αντιδράσεις όπως αύξηση προσοχής και κατάσταση εγρήγορσης,
- να διευκολύνει στην ομαδοποίηση, αλλά και στο διαχωρισμό στοιχείων της εφαρμογής,
- να δίνει έμφαση στη λογική οργάνωση της εφαρμογής,
- να προσθέτει ενδιαφέρον στην εφαρμογή,
- να βελτιώνει την επίδοση του χρήστη σε μια σειρά από επαναλαμβανόμενες λειτουργίες.

Το χρώμα που χρησιμοποιείται στο παρασκήνιο είναι το βασικό χρώμα για την εμφάνιση της οθόνης. Αντίθετα το χρώμα προσκήνιου είναι το βασικό χρώμα για την εμφάνιση της πληροφορίας. Τα χρώματα που χρησιμοποιούνται στο προσκήνιο και στο παρασκήνιο πρέπει να διατηρούν κάποια αντίθεση. Μερικά από τα προτεινόμενα ζεύγη χρωμάτων για το προσκήνιο και το παρασκήνιο φαίνονται στον πίνακα 12.1.

Πίνακας 12.1 : Προτεινόμενοι συνδυασμοί παρασκηνίου και προσκήνιου

Παρασκήνιο	Προσκήνιο
Άσπρο	Μπλε
Μπεζ	Μαύρο ή σκούρο μπλε
Ανοικτό γκρι	Μαύρο
Μαύρο	Άσπρο
Μπλε	Άσπρο

Στην προσπάθεια επιλογής σωστού συνδυασμού χρωμάτων για το προσκήνιο και το παρασκήνιο θα πρέπει να έχουμε υπόψη μας την ανάγκη του χρήστη να ξεχωρίζει ανάμεσα στο προσκήνιο και στο παρασκήνιο και να αντιλαμβάνεται σωστά το ανθρώπινο μάτι τα δύο επίπεδα. Μπλε κείμενο προσκηνίου σε κόκκινο παρασκήνιο δεν είναι σωστός συνδυασμός γιατί το κόκκινο φαίνεται σαν να είναι μπροστά και να περιβάλλει το μπλε κείμενο.

Μερικές οδηγίες για τη σωστή χρήση του χρώματος είναι οι εξής:

- *Συντηρητική χρήση*. Η υπερβολική χρήση χρωμάτων μπορεί να έχει αρνητικό αποτέλεσμα. Η χρήση τους πρέπει να γίνεται στο μέτρο που απαιτείται Αν εμφανίσουμε στην οθόνη μια λέξη ή μια πρόταση γραμμένη με συνδυασμό πολλών χρωμάτων, το αποτέλεσμα μπορεί να είναι εντυπωσιακό, ιδιαίτερα αν κοιτάμε την οθόνη από μακριά, όμως από κοντά είναι πολύ πιθανό ότι θα υπάρχει δυσκολία ανάγνωσης του συγκεκριμένου κειμένου.
- *Περιορισμένος αριθμός χρωμάτων*. Ένας κανόνας που μπορεί αν εφαρμοστεί είναι η χρήση μέχρι τεσσάρων χρωμάτων για παρουσίαση κειμένου, και μέχρι επτά για παρουσίαση που περιλαμβάνει και γραφικά
- *Αναγνώριση του χρώματος ως τεχνική κωδικοποίησης*. Με το χρώμα αναγνωρίζονται πολύ γρήγορα κάποιες διεργασίες που επιτελούνται. Για το λόγο αυτό θα πρέπει να δίνεται προσοχή στην κωδικοποίηση του χρώματος και να τηρούνται οι γενικά αποδεκτές χρωματικές κωδικοποιήσεις. Για παράδειγμα το κόκκινο σημαίνει απαγόρευση ή παύση. Αν ένας σχεδιαστής το χρησιμοποιεί για να τονίσει ένα μήνυμα, το πιθανότερο αποτέλεσμα θα είναι να δημιουργηθεί σύγχυση στο χρήστη.

- *Σταθερότητα στην κωδικοποίηση*. Αν, για παράδειγμα, ένα μήνυμα προειδοποίησης εμφανίζεται με κίτρινο χρώμα σε κάποιο σημείο της εφαρμογής, τότε θα πρέπει να εμφανίζονται με το ίδιο χρώμα όλα τα μηνύματα παρόμοιου τύπου που τυχόν υπάρχουν στην εφαρμογή.
- *Χρήση προς βοήθεια της μορφοποίησης*. Σε απεικονίσεις που είναι πολύ πυκνές και όπου ο χώρος είναι πολύτιμος, παρόμοια χρώματα μπορούν να χρησιμοποιηθούν για να ομαδοποιήσουν συσχετιζόμενα αντικείμενα.
- *Συσχέτιση με το ακροατήριο στο οποίο απευθύνεται*. Ο σχεδιαστής της εφαρμογής πρέπει να λαμβάνει υπόψη του το ακροατήριο στο οποίο απευθύνεται η εφαρμογή και να επιλέγει τα καταλληλότερα χρώματα και με βάση τις χρωματικές προτιμήσεις του. Έρευνες αναδεικνύουν τα χρώματα που προτιμούν διαφορετικές ηλικίες χρηστών. Στον πίνακα 12.2 παρουσιάζονται οι χρωματικές προτιμήσεις παιδιών και εφήβων.

Πίνακας 12.2 : Τα περισσότερο δημοφιλή χρώματα κατά σειρά προτίμησης

Παιδιά	Έφηβοι
Κίτρινο	Μπλε
Άσπρο	Κόκκινο
Ροζ	Πράσινο
Κόκκινο	Άσπρο
Μπλε	Ροζ
Πράσινο	Μοβ
Μοβ	Κίτρινο

Η πυκνότητα οθόνης αναφέρεται ως μέτρο του συνόλου της πληροφορίας που εμφανίζεται στο ενεργό τμήμα της οθόνης.

12.4.2. Μηνύματα λάθους

Τα μηνύματα λαθών πρέπει να είναι όσο το δυνατόν πιο κατατοπιστικά για να μπορούν να προσφέρουν ουσιαστική βοήθεια στους αρχάριους χρήστες. Όταν τα μηνύματα δεν είναι κατανοητά, τότε οι χρήστες μπερδεύονται περισσότερο, αγχώνονται και εγκαταλείπουν την προσπάθεια χρήσης του λογισμικού. Βελτιώνοντας τα μηνύματα λαθών, βελτιώνουμε ταυτό-

χρονα και την απόδοση χρήσης του λογισμικού. Η μορφή και το περιεχόμενο των μηνυμάτων λαθών επηρεάζουν την απόδοση του χρήστη και λειτουργούν σαν κίνητρο για τη συνέχιση της προσπάθειας παρά το γεγονός ότι έχει κάνει λάθος.

Βασικές αρχές για τη σχεδίαση των μηνυμάτων λαθών, που μπορούν αν αποβούν χρήσιμες για την υποβοήθηση τόσο αρχάριων, αλλά και έμπειρων χρηστών, είναι :

- *Να είναι εξειδικευμένα*. Τα μηνύματα που είναι πολύ γενικά δεν προσφέρουν ουσιαστική βοήθεια στο χρήστη έτσι ώστε να καταλάβει το είδος του λάθους που έκανε.

Το μήνυμα "Άνοιγμα λάθους αρχείου", δεν προσφέρει κάποια ουσιαστική και χρήσιμη πληροφορία. Αντίθετα αν εμφανιζόταν το μήνυμα "Μόνο αρχεία μορφοποίησης εικόνας μπορείς να ανοίξεις", η πληροφορία που παρέχεται το χρήστη είναι κατάλληλη και εποικοδομητική.

- *Να καθοδηγούν το χρήστη*. Εκτός από το να ενημερώνουμε τους χρήστες για το λάθος τους, θα πρέπει να τους δίνουμε και κάποιες οδηγίες σχετικά με τον τρόπο που θα το διορθώσουν.

Αντί να εμφανίζεται το μήνυμα "Μη ορισμένα αρχεία εξόδου" θα μπορούσε να εμφανίζεται το μήνυμα "Όρισε τα αρχεία στα όποια θέλεις να καταγραφούν τα αποτελέσματα της επεξεργασίας".

Τα σύγχρονα περιβάλλοντα προγραμματισμού παρέχουν τυποποιημένα πλαίσια μηνυμάτων που εμφανίζονται με σταθερό σχήμα και μορφοποίηση.

- *Να έχουν θετικό τόνο*. Να ορίζουν τι πρέπει να γίνει παρά να επικρίνουν και να αμφισβητούν το χρήστη. Όροι όπως "Κακός χειρισμός", "Άστοχη χρήση" πρέπει να αποφεύγονται.
- *Να έχουν σταθερή μορφοποίηση*. Τα μηνύματα λάθους μπορούν να γράφονται άλλοτε με κεφαλαία και άλλοτε με πεζά γράμματα Τα κεφαλαία γράμματα προτείνεται να χρησιμοποιούνται σε μηνύματα που είναι σύντομα και εμπεριέχουν έννοιες κινδύνου ή προειδοποίησης. Το μέγεθος των πλαισίων μηνυμάτων πρέπει να είναι σταθερό και η εμφάνισή τους να γίνεται στο ίδιο πάντοτε σημείο.

12.5. Ηχητική σχεδίαση της διεπαφής χρήστη

Η χρήση του ήχου στα σύγχρονα προγραμματιστικά περιβάλλοντα είναι ευρέως διαδεδομένη. Το να χρησιμοποιήσει κάποιος δεδομένα ήχου σε μια εφαρμογή, τεχνολογικά δεν είναι ιδιαίτερα πολύπλοκο. Ο ήχος δίνει μία

άλλη διάσταση στην εφαρμογή, αφού μπορεί να αποβεί χρήσιμος τόσο λειτουργικά, όσο και από άποψη περιεχομένου.

Ο ήχος αποτελεί μια από τις πολυμεσικές μορφές δεδομένων. Χρήση ήχου σε μια εφαρμογή, μπορεί να γίνει με τρεις διαφορετικούς τρόπους:

- *Αφήγηση*: Η αφήγηση θα πρέπει να περιγράφει περισσότερα από όσα μπορεί να δει και να καταλάβει ο χρήστης από τα οπτικά δεδομένα της εφαρμογής. Η καλά σχεδιασμένη αφήγηση συμπληρώνει την πληροφορία που παρέχει το οπτικό μέρος της εφαρμογής. Δίνει προσοχή σε σημαντικά σημεία που διαφορετικά τυχόν θα αγνοηθούν.

Η αφήγηση πρέπει να συνεργάζεται χρονικά σωστά με τα οπτικά δεδομένα που συνοδεύει. Θα πρέπει να γίνει σωστή επιλογή των λέξεων που περιγράφουν πιστότερα τις έννοιες που σκοπός είναι να παρουσιαστούν. Η επιλογή αυτή θα πρέπει να γίνει με προσοχή, ώστε οι λέξεις που θα χρησιμοποιηθούν στην αφήγηση να έχουν πραγματική σχέση με την οπτική παρουσίαση της εφαρμογής.

Ο τόνος της φωνής του αφηγητή παίζει σημαντικό ρόλο. Η αίσθηση και η διάθεση που μπορεί να μεταδώσει ο τόνος της φωνής, είναι το ίδιο σημαντική όσο και το περιεχόμενο της αφήγησης. Ο τόνος της αφήγησης δεν θα πρέπει να συγκρούεται με το μήνυμα. Ένα σοβαρό θέμα μπορεί να εκμηδενιστεί από μια αφήγηση που είναι αδιάφορη ή ιδιότροπη. Θα πρέπει να τονιστεί ότι το ίδιο κείμενο εκφωνούμενο από δύο διαφορετικούς αφηγητές, μπορεί να επιφέρει διαφορετικά αποτελέσματα στο ίδιο κοινό.

- *Μουσική επένδυση*: Η μουσική είναι ένα από τα λίγα πράγματα που δεν απαιτεί μετάφραση, που παρουσιάζει μια παγκοσμιότητα. Η μουσική υπόκρουση σε μία εφαρμογή μπορεί να προκαλέσει μεταβολή της συναισθηματικής κατάστασης του χρήστη. Μπορεί να τον θέσει σε εγρήγορση, να του μεταδώσει γαλήνη, να τον συναρπάσει.

Η χρήση της μουσικής υπόκρουσης δεν θα πρέπει να αποτελεί αυτοσκοπό σε μια εφαρμογή, αλλά μέρος της λειτουργικότητάς της και του περιεχομένου που θέλει να παρουσιάσει στο χρήστη. Η ένταση της μουσικής υπόκρουσης θα πρέπει να είναι τέτοια που να μη δημιουργεί ενόχληση, ούτε να δυσχεραίνει την αφήγηση που τυχόν συνυπάρχει.

- *Ηχητικά σήματα*: Οι ηχητικές αντιδράσεις του λογισμικού σε κάποιες ενέργειες του χρήστη παίζουν σημαντικό ρόλο τραβώντας την προσοχή του. Η χρήση ηχητικών σημάτων μπορεί να γίνει με αμφίδρομο κίνητρο. Δηλαδή, είτε όταν ο χρήστης προκαλέσει μια ενέργεια στην οποία κρίνεται ότι το σύστημα πρέπει να "απαντήσει", είτε όταν το σύστημα εκτελέσει μια διαδικασία για την οποία θα πρέπει να ενημερωθεί ο χρή-

στης. Ηχητικά σήματα χρησιμοποιούνται συνήθως σε περιπτώσεις όπου ο χρήστης προχωρεί σε λανθασμένους χειρισμούς, μη αναστρέψιμες ενέργειες κλπ. ή όταν ολοκληρώνεται (στο προσκήνιο ή και στο παρασκήνιο) η εκτέλεση κάποιας εργασίας για την οποία ο χρήστης θα πρέπει να λάβει γνώση.

Ανακεφαλαίωση

Το κεφάλαιο αυτό πραγματεύτηκε το θέμα της διεπαφής χρήση (user interface) από την πλευρά του λογισμικού. Αναλύθηκε ο όρος και περιγράφτηκαν τα είδη διεπαφής χρήστη, με έμφαση στη γραφική διεπαφή χρήστη (GUI). Παρουσιάστηκαν οι βασικές γενικές αρχές σχεδίασης διεπαφής χρήστη, όπως και οι φάσεις ανάπτυξής της. Στη συνέχεια το ενδιαφέρον επικεντρώθηκε σε θέματα σχεδίασης που αφορούν βασικές συνιστώσες του οπτικού μέρους της διεπαφής χρήστη. Τέλος το κεφάλαιο ολοκληρώνεται με αναφορά σε θέματα που σχετίζονται με το ηχητικό μέρος της διεπαφής χρήστη.

Λέξεις κλειδιά

Διεπαφή χρήστη (user interface), εργονομία λογισμικού, σχεδίαση διεπαφής χρήστη, χρώμα, μηνύματα λάθους

Ερωτήσεις - Θέματα για συζήτηση

1. Να γίνει συζήτηση σχετικά με τις διαφορετικές αποδόσεις του αγγλικού όρου user interface και να εντοπιστεί ο καταλληλότερος σύμφωνα με την άποψη των μαθητών την οποία και πρέπει να τεκμηριώσουν.
2. Να γίνει συζήτηση σχετικά με τα πλεονεκτήματα χρήσης των γραφικών περιβαλλόντων.
3. Να αναφέρουν οι μαθητές τα σπουδαιότερα χαρακτηριστικά του GUI.
4. Να περιγράψουν οι μαθητές τις φάσεις ανάλυσης διεπαφής χρήστη.
5. Να γίνει συζήτηση σχετικά με τις προσωπικές νοηματικές συνδέσεις των μαθητών ανάμεσα σε χρώματα και καταστάσεις.
6. Να περιγράψουν οι μαθητές τις βασικές αρχές παρουσίασης μηνυμάτων λαθών.

7. Ν' απαντήσουν οι μαθητές στο ερώτημα "Για ποιους λόγους θα συμπεριλαμβάνατε ηχητικά σε μια εφαρμογή μισθοδοσίας και ποια θα ήταν αυτά;".

Βιβλιογραφία

1. Lon Brafield: The user interface – Concepts & Design, Addison-Wesley Publishing Company, Amsterdam, Holland,1993.
2. Robert Lindstrom: Multimedia Presentations – Create dynamic presentations that inspire, Osborne-McGraw Hill, USA, 1994.
3. Arch Luther: Designing Interactive Multimedia, Bantam Books, New York, USA, 1992.
4. Mark Maubury: Intelligent Multimedia Interfaces, AAAIPress/The MIT Press, California, USA, 1993.
5. Karen McGraw: Designing and Evaluating User Inetrfaces for Knowledge_based Systems, Ellis Horwood Limite, West Sussex, England, 1992.
6. Παναγιώτης Πολίτης: Υπερκείμενα, υπερμέσα και πολυμέσα, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1994.
7. Ben Shneiderman: Designing the User Interface Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company Inc, New York, USA, 1992.

Διευθύνσεις Διαδικτύου

http://cfg.cit.cornell.edu/design/contents.html

Ιστοσελίδα με ενδιαφέρουσες αναφορές σε μια σειρά από θέματα σχετικά με τη διεπαφή χρήστη, όπως κύριες έννοιες, η διαδικασία της σχεδίασής της, τα βασικά δομικά στοιχεία που χρησιμοποιούνται, κ.λπ. Περιλαμβάνει τέλος και σχετική επιπρόσθετη βιβλιογραφία.

http://www.sju.edu/~jhodgson/gui/guihome.html

Σε αυτήν την ιστοσελίδα μπορείτε να βρείτε χρήσιμα στοιχεία σχετικά με τη σχεδίαση της διεπαφής χρήστη. Ουσιαστικά πρόκειται για μια παρουσίαση δομημένη σε επτά ξεχωριστές ενότητες, που αποτελούν από σχετικά απλά μέχρι αρκετά σύνθετα υποθέματα της σχεδίασης διεπαφής χρήστη.

ΚΕΦΑΛΑΙΟ 13ο

ΕΚΣΦΑΛΜΑΤΩΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΠΕΡΙΕΧΟΜΕΝΑ

- Κατηγορίες λαθών
- Εκσφαλμάτωση
- Εργαλεία εκσφαλμάτωσης
- Χειρισμός λαθών κατά το χρόνο εκτέλεσης

Εισαγωγή

Ανεξάρτητα από το πόσο προσεκτικά έχει γράψει κάποιος ένα πρόγραμμα, τις περισσότερες φορές απρόβλεπτες καταστάσεις, κακός χειρισμός ή λογικά λάθη στο σχεδιασμό των προγραμμάτων, οδηγούν στην εμφάνιση λαθών. Τα λάθη που παρουσιάζονται κατά την εκτέλεση ενός προγράμματος, μερικές φορές μπορεί να είναι ασήμαντα, όπως για παράδειγμα ή εσφαλμένη στοίχιση των αποτελεσμάτων. Άλλες φορές όμως μπορεί να είναι ιδιαίτερα κρίσιμα, ώστε να οδηγούν στην κατάρρευση των εφαρμογών ή του συστήματος. Ένα πρόγραμμα πριν παραδοθεί για πραγματική λειτουργία, πρέπει να ελέγχεται και να είναι βέβαιο ότι εργάζεται απρόσκοπτα και παράγει σωστά αποτελέσματα. Κάθε προγραμματιστής οφείλει κατά τη φάση της υλοποίησης, να δοκιμάζει λεπτομερώς το πρόγραμμα σε διαφορετικές συνθήκες και με ποικιλία δεδομένων, να διορθώνει όποια λάθη παρουσιαστούν και να συγκρίνει τα παραγόμενα αποτελέσματα με τα αναμενόμενα, ώστε να προλαμβάνει απρόσμενες καταστάσεις λάθους, πριν εμφανιστούν σε πραγματική λειτουργία.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να αναγνωρίζουν ένα λάθος.
- να ορίζουν τις κατηγορίες λάθους.
- να αναφέρουν εργαλεία εκσφαλμάτωσης και τον τρόπο χειρισμού τους.
- να εφαρμόζουν τις τεχνικές χειρισμού λαθών κατά το χρόνο εκτέλεσης.

Προερωτήσεις

- ✓ ποια είναι τα πιθανά λάθη που μπορεί να εμφανιστούν κατά την υλοποίηση και την εκτέλεση ενός προγράμματος;
- ✓ ποια είναι τα πλεονεκτήματα της εκσφαλμάτωσης;
- ✓ υπάρχουν εργαλεία που βοηθούν τον προγραμματιστή για την εκσφαλμάτωση ενός προγράμματος;
- ✓ είναι εφικτό να χειριστούμε αναπάντεχες καταστάσεις που πιθανόν θα εμφανιστούν κατά το χρόνο εκτέλεσης του προγράμματος, έτσι ώστε να μην διακόπτεται η λειτουργία του;
- ✓ η εργασία της εκσφαλμάτωσης θεωρείται ότι είναι απαραίτητη για την υλοποίηση ενός προγράμματος;

13.1 Κατηγορίες λαθών

Ένας προγραμματιστής, ανεξάρτητα από πόσο ικανός είναι, όταν δημιουργεί ένα πρόγραμμα, είναι φυσικό να κάνει κάποιο λάθος. Ειδικά σε μεγάλες εφαρμογές που κατασκευάζει πολύπλοκες υπολογιστικές ρουτίνες ή χρησιμοποιεί αρκετές συσκευές υλικού, είναι αδύνατο να αποφύγει τέτοιες ανεπιθύμητες καταστάσεις.

Σε ένα πρόγραμμα είναι δυνατό να παρουσιαστούν διαφορετικής μορφής λάθη, τα οποία μπορούν να χωριστούν σε τρεις βασικές κατηγορίες:

Λάθη κατά την υλοποίηση

Τα λάθη κατά το χρόνο υλοποίησης, προκαλούνται κυρίως από λανθασμένη σύνταξη εντολών προγράμματος. Τέτοια λάθη μπορεί να είναι η λανθασμένη συγγραφή μιας δεσμευμένης λέξης της γλώσσας προγραμματισμού ή η χρήση μιας δομής ελέγχου χωρίς την εντολή τερματισμού της.

Ένα λάθος που προκαλείται κατά τη συγγραφή του προγράμματος, ανιχνεύεται από το μεταγλωττιστή, ο οποίος εμφανίζει προς το προγραμματιστή κάποιο προειδοποιητικό μήνυμα. Αν το πρόγραμμα περιέχει ένα λάθος αυτής της μορφής, δεν επιτρέπεται η εκτέλεσή του, μέχρι να το διορθώσει ο προγραμματιστής.

Τα σύγχρονα προγραμματιστικά περιβάλλοντα μας προφυλάσσουν αυτόματα από τα λάθη κατά την υλοποίηση , αφού παρέχουν εργαλεία αυτόματου ελέγχου σύνταξης των εντολών και παρακολουθούν τον προγραμματιστή κατά τη συγγραφή του προγράμματος. Μόλις διαπιστώσουν κάποιο συντακτικό λάθος, σταματούν και απαιτούν τη διόρθωσή του. Συνήθως αντιλαμβάνονται ακριβώς το λάθος που δημιουργήθηκε και προτείνουν αναλυτικά τον τρόπο διόρθωσής του, εμφανίζοντας σε ενημερωτικό πλαίσιο την ορθή σύνταξη της εντολής που προκλήθηκε το λάθος.

*Μια από τις πρώτες βλάβες που εμφανίστηκαν στους υπολογιστές ήταν ένα βραχυκύκλωμα που οφειλόνταν στον εγκλωβισμό ενός εντόμου (bug) στο εσωτερικό ενός υπολογιστή. Από το γεγονός αυτό, είναι πολύ πιθανό να προέρχεται ο όρος **bugs**, με τον οποίο αναφερόμαστε στα λάθη που εμφανίζονται σε ένα πρόγραμμα.*

Λάθη κατά την εκτέλεση

Τα λάθη που προκαλούνται κατά το χρόνο εκτέλεσης του προγράμματος, είναι πιο επώδυνα γιατί συνήθως εμφανίζονται σε πραγματικό περιβάλλον εκτέλεσης και τις περισσότερες φορές προκαλούν τον αντικανονικό τερματισμό της εφαρμογής και το *κρέμασμα* (crash) του συστήματος.

Όταν ένα λάθος προκληθεί κατά την εκτέλεση της εφαρμογής, είναι δυνατό να αντιμετωπισθεί μόνο με τη χρήση εντολών προγράμματος που το παγιδεύουν και εκτελούν τις κατάλληλες διαδικασίες χειρισμού του.

Η πρόληψη τέτοιων λαθών είναι αρκετά δύσκολη, αφού συνήθως οφείλονται σε καταστάσεις που δεν είναι εύκολο να ελεγχθούν από τον προγραμματιστή, ενώ πολλές φορές εμφανίζονται μετά από μεγάλο χρονικό διάστημα. Τέτοια λάθη είναι δυνατό να προκληθούν από την κλήση μιας διαδικασίας με δεδομένα που δεν μπορεί να χειριστεί, όπως η αναζήτηση διαγραμμένων αρχείων, η προσπάθεια διαίρεσης ενός αριθμού με το μηδέν, η υπερχείλιση μιας αριθμητικής μεταβλητής ή από δυσλειτουργία του υλικού μέρους του υπολογιστή, όπως η καταστροφή του σκληρού δίσκου του συστήματος, ο τερματισμός μιας σύνδεσης δικτύου και η αποσύνδεση του εκτυπωτή.

Λογικά λάθη

Τα λογικά λάθη είναι συνήθως λάθη σχεδιασμού και δεν προκαλούν τη διακοπή της εκτέλεσης του προγράμματος. Ενώ ο μεταγλωττιστής της γλώσσας προγραμματισμού δεν ανιχνεύει κανένα συντακτικό λάθος και κατά την εκτέλεση του προγράμματος δεν παρουσιάζονται ανεπιθύμητες καταστάσεις σφαλμάτων, τελικά δεν παράγονται τα επιθυμητά αποτελέσματα.

Η ανίχνευση τέτοιων λαθών δεν είναι δυνατό να πραγματοποιηθεί από κάποιο εργαλείο του υπολογιστή και διαπιστώνονται μόνο με τη *διαδικασία ελέγχου* (testing) και την ανάλυση των αποτελεσμάτων των προγραμμάτων.

Το πιο δημοφιλές λάθος που παρουσιάστηκε στην ιστορία των υπολογιστών είναι το **πρόβλημα του έτους 2000** (millennium bug). Το πρόβλημα αυτό είναι ιδιόμορφο, γιατί οφείλεται σε συνδυασμένη προβληματική λειτουργία του λογισμικού και του υλικού μέρους του υπολογιστή και ακόμη απασχόλησε χρονικά την κοινωνία μας αρκετά πριν από την ουσιαστική εμφάνιση των συνεπειών του.

Άλλα σημαντικά προβλήματα που παρουσιάστηκαν στην ιστορία των υπολογιστών και αποδόθηκαν στη δυσλειτουργία του λογισμικού είναι :

- Το 1962 το διαστημικό όχημα Mariner 1 εκτοξεύτηκε από το ακρωτήρι Canaveral των Ηνωμένων Πολιτειών της Αμερικής για τον πλανήτη Αφροδίτη. Μετά την απογείωση ο πύραυλος που κατεύθυνε το διαστημικό όχημα έχασε την κατεύθυνσή του και οι τεχνικοί της διαστημικής υπηρεσίας των Η.Π.Α. (NASA) αναγκάστηκαν να τον ανατινάξουν, πριν αυτός καταστραφεί σε κάποιο σημείο της γης και κινδυ-

νεύσουν ανθρώπινες ζωές. Ευτυχώς το διαστημικό όχημα δεν είχε ανθρώπινο πλήρωμα. Η αναφορά της NASA απέδωσε το ατύχημα στην εσφαλμένη αντικατάσταση ενός θετικού πρόσημου με αρνητικό σε μια εντολή FORTRAN του λογισμικού πλοήγησης του πυραύλου. Το εσφαλμένο πρόσημο στοίχισε περίπου 80 εκατομμύρια δολάρια.

- Το 1990 ένα λάθος σε μια εντολή του λογισμικού των νέων συστημάτων δρομολόγησης τηλεφωνικών κλήσεων της εταιρείας τηλεπικοινωνιών AT&T, προκάλεσε το μπλοκάρισμα του μεγαλύτερου τμήματος του τηλεφωνικού δικτύου των Ηνωμένων Πολιτειών της Αμερικής. Περίπου 5 εκατομμύρια τηλεφωνικές γραμμές διακόπηκαν για εννέα ώρες.

- Ο επεξεργαστής Pentium το έτος 1994 παρουσίασε δυσλειτουργία, αφού σε σπάνιες περιπτώσεις έδινε εσφαλμένες απαντήσεις σε πολύπλοκες μαθηματικές εξισώσεις. Το πρόβλημα ανακαλύφτηκε από τον καθηγητή Thomas Nicely στο Πανεπιστήμιο Lynchburg στη Virginia των Ηνωμένων Πολιτειών της Αμερικής. Αρχικά η κατασκευάστρια εταιρεία αρνήθηκε την ύπαρξή του, αλλά στη συνέχεια υποχρεώθηκε να αντικαταστήσει τους προβληματικούς επεξεργαστές και σύμφωνα με εκτιμήσεις δαπάνησε γι αυτό το σκοπό περίπου 450 εκατομμύρια δολάρια

- Το έτος 1995 επρόκειτο να γίνουν τα εγκαίνια του νέου διεθνούς αεροδρομίου στο Denver των Ηνωμένων Πολιτειών της Αμερικής. Το αεροδρόμιο είχε κατασκευαστεί με σύγχρονη τεχνολογία και διέθετε ένα αυτόματο σύστημα μεταφοράς αποσκευών. Με την έναρξη λειτουργίας του το σύστημα μεταφοράς αποσκευών παρουσίασε λειτουργικά προβλήματα, με συνέπεια την καταστροφή αποσκευών επιβατών και του ιδίου του συστήματος. Το αεροδρόμιο διέκοψε τη λειτουργία του και επαναλειτούργησε δεκαέξι μήνες αργότερα και με κλασικό σύστημα μεταφοράς αποσκευών, επιβαρύνοντας τον προϋπολογισμό κατασκευής του κατά 3,2 εκατομμύρια δολάρια.

13.2 Εκσφαλμάτωση

Η διαδικασία ελέγχου, εντοπισμού και διόρθωσης των σφαλμάτων ενός προγράμματος καλείται *εκσφαλμάτωση* (debugging). Στόχος της διαδικασίας εκσφαλμάτωσης είναι ο εντοπισμός των σημείων του προγράμματος που προκαλούν προβλήματα στη λειτουργία του.

Η εργασία της εκσφαλμάτωσης δεν είναι εύκολη, απαιτεί βαθιά γνώση της γλώσσας προγραμματισμού και φυσικά αντίστοιχες ικανότητες από τον προγραμματιστή. Για τον εντοπισμό ενός λάθους δεν υπάρχουν ιδιαίτερα μυστικά και τρυκ. Η εκσφαλμάτωση είναι ένα πρόβλημα λογικής και όσο πιο καλά αντιλαμβάνεται ο προγραμματιστής τον τρόπο που εργάζεται το πρόγραμμα, τόσο πιο εύκολα και σύντομα θα εντοπίσει λάθη που προκαλούν δυσλειτουργίες.

Η εισαγωγή γραμμών με σχόλια σε ένα πρόγραμμα υποβοηθά σημαντικά την εκσφαλμάτωση.

Σε ένα σύγχρονο προγραμματιστικό περιβάλλον δεν χρειάζεται ιδιαίτερα μνεία για τα λάθη που παρουσιάζονται κατά το χρόνο σχεδιασμού, αφού αυτά, όπως αναφέρθηκε, είναι συντακτικά λάθη και τις περισσότερες φορές το περιβάλλον προγραμματισμού τα ανιχνεύει αυτόματα και προτείνει τη διόρθωσή τους. Ακόμη και αν το περιβάλλον δεν προτείνει τη διόρθωση, ο μεταγλωττιστής συλλαμβάνει και περιγράφει το λάθος και στη συνέχεια ο προγραμματιστής μπορεί πολύ εύκολα να το διορθώσει.

Τα λάθη που κυρίως μας απασχολούν στη φάση της εκσφαλμάτωσης είναι τα λογικά λάθη και τα λάθη που παρουσιάζονται κατά το χρόνο εκτέλεσης του προγράμματος. Η εκσφαλμάτωση τέτοιων λαθών μπορεί να γίνει μέσα από εργαλεία εκσφαλμάτωσης ή από ειδικές εντολές ή συναρτήσεις που προσφέρει το περιβάλλον προγραμματισμού.

13.3. Εργαλεία εκσφαλμάτωσης

Τα ονόματα των μεταβλητών πρέπει να ανάγουν στο περιεχόμενό τους. Έτσι διευκολύνεται η εκσφαλμάτωση.

Πολλές φορές ο εντοπισμός ενός σφάλματος είναι ιδιαίτερα δύσκολος και για να επιτευχθεί χρειάζεται συστηματική παρακολούθηση και ανάλυση των δεδομένων του προγράμματος. Τα σύγχρονα εργαλεία προγραμματισμού δίνουν τη δυνατότητα στον προγραμματιστή να παρακολουθεί, τι συμβαίνει στο παρασκήνιο του προγράμματος κατά το χρόνο εκτέλεσης. Αυτό επιτυγχάνεται με τη χρήση κατάλληλων εργαλείων, που παρέχουν πολλές δυνατότητες ελέγχου κατά τη δοκιμαστική εκτέλεση ενός προγράμματος. Τα περισσότερα εργαλεία προγραμματισμού συνοδεύονται από *προγράμματα διόρθωσης* (debuggers).

Με τα εργαλεία εκσφαλμάτωσης είναι δυνατό να εμφανίζεται η τιμή μιας μεταβλητής ή μιας έκφρασης, να γίνεται δυναμική επέμβαση σε κάποιο σημείο δίνοντας διαφορετική τιμή σε μια μεταβλητή ή να εκτελείται βήμα προς βήμα το πρόγραμμα για τον εντοπισμό της εντολής που προκαλεί το σφάλμα.

Στη συνέχεια θα αναφέρουμε τις βασικές λειτουργίες εκσφαλμάτωσης που προσφέρουν τα περισσότερα προγραμματιστικά περιβάλλοντα :

Εκφράσεις ελέγχου

Οι *εκφράσεις ελέγχου* (watch expressions) είναι χρήσιμες για την άμεση παρατήρηση τιμών μεταβλητών ή εκφράσεων κατά την εκτέλεση ενός προγράμματος. Ακόμη με μια έκφραση ελέγχου, υπάρχει η δυνατότητα να διακοπεί η εκτέλεση ενός προγράμματος, όταν αλλάξει το περιεχόμενο μιας μεταβλητής ή δεχτεί συγκεκριμένη τιμή. Με τις εκφράσεις ελέγχου παρατηρούμε τη συμπεριφορά μιας μεταβλητής ή μιας έκφρασης σε όλη τη διάρκεια εκτέλεσης του προγράμματος.

Εκφράσεις ελέγχου

Expression	Value	Type	Context
Int(varA) + Int(varB)	12	Variant/Double	FrmDebug.Command1_Click
varA	"4"	Variant/String	FrmDebug.Command1_Click
varB	"8"	Variant/String	FrmDebug.Command1_Click

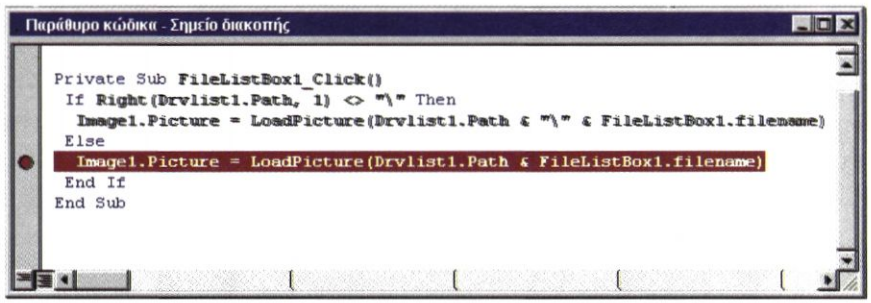
Σχ. 13.1. *Εμφάνιση εκφράσεων ελέγχου*

Σημεία διακοπής

Ένα *σημείο διακοπής* (breakpoint) διακόπτει την εκτέλεση ενός προγράμματος ακριβώς πριν από μια συγκεκριμένη εντολή. Τα σημεία διακοπής χρησιμοποιούνται για τον έλεγχο της κατάστασης των δεδομένων σε συγκεκριμένο σημείο του προγράμματος. Ορίζονται στον πηγαίο κώδικα με το μαρκάρισμα κάθε εντολής, στην οποία θέλουμε να διακόψουμε την εκτέλεση του προγράμματος.

Βήμα προς βήμα εκτέλεση

Μετά τη διακοπή εκτέλεσης από ένα σημείο διακοπής, παρέχεται η δυνατότητα να συνεχιστεί βήμα προς βήμα η εκτέλεση του προγράμματος. Με τη λειτουργία αυτή μπορούμε να εξετάσουμε βηματικά τα αποτελέσματα κάθε εντολής ή ρουτίνας του προγράμματος.



Σχ. 13.2. *Σημείο διακοπής στον κώδικα της εφαρμογής*

Ιστορικό

Με αυτή τη λειτουργία κατακρατείται *ιστορικό* (history) των τελευταίων εντολών που εκτελέστηκαν. Διαβάζοντας το ιστορικό αυτό μπορούμε να ελέγξουμε τη ροή του προγράμματος βαδίζοντας μπρος ή πίσω μέσα στον κώδικα.

Ιχνηλάτηση

Δίνει τη δυνατότητα αργής εκτέλεσης του προγράμματος, ενώ παράλληλα εμφανίζεται φωτισμένη η εντολή που εκτελείται κάθε φορά. Η *ιχνηλάτηση* (tracing) ενός προγράμματος ξεκινά από ένα σημείο διακοπής, που έχει εισαχθεί στο πρόγραμμα και λειτουργεί σε επίπεδο εντολής εκτελώντας κάθε φορά μια εντολή ή σε επίπεδο διαδικασίας εκτελώντας κάθε φορά όλη τη διαδικασία.

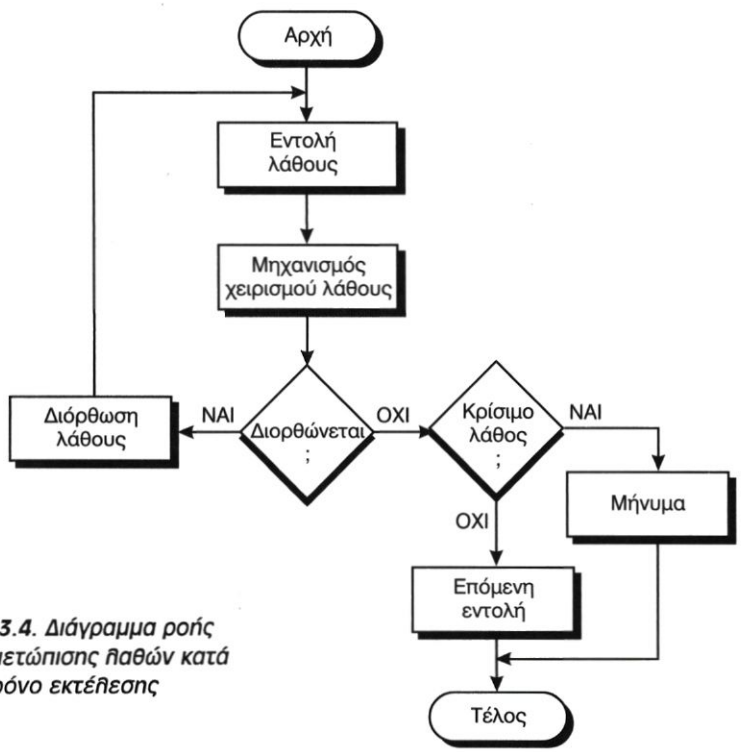
Λειτουργία άμεσης εκτέλεσης

Με τη λειτουργία *άμεσης εκτέλεσης* έχουμε τη δυνατότητα σε κάποιο σημείο διακοπής να πληκτρολογήσουμε και να εκτελέσουμε άμεσα οποιαδήποτε εντολή με σκοπό τη λογική αλλαγή της ροής εκτέλεσης του προγράμματος.

Σχ. 13.3. *Εμφάνιση των περιεχομένων των εκφράσεων σε λειτουργία άμεσης εκτέλεσης*

Οι λειτουργίες των εργαλείων εκσφαλμάτωσης σε γραφικά περιβάλλοντα πραγματοποιούνται μέσα από ειδικά παράθυρα (π.χ. παράθυρο ελέγχου, παράθυρο άμεσης εκτέλεσης), ενώ στα υπόλοιπα εργαλεία από επιλογές μενού επιλογών που προσφέρει το περιβάλλον ανάπτυξης ή με ειδικές εντολές στον πηγαίο κώδικα.

Ένας προγραμματιστής που δεν είναι εξοικειωμένος και δεν γνωρίζει το χειρισμό των εργαλείων εκσφαλμάτωσης ή αν εργάζεται σε ένα περιβάλλον που δεν του παρέχει αντίστοιχα εργαλεία, μπορεί να υλοποιήσει αρκετές από τις λειτουργίες εκσφαλμάτωσης που αναφέρθηκαν, με την παρεμβολή εντολών εμφάνισης ή εκτύπωσης μηνυμάτων, τιμών μεταβλητών ή εκφράσεων.



Σχ. 13.4. Διάγραμμα ροής αντιμετώπισης λαθών κατά το χρόνο εκτέλεσης

13.4 Χειρισμός λαθών κατά το χρόνο εκτέλεσης

Τα λάθη που προκαλούνται κατά την εκτέλεση ενός προγράμματος μπορούν να προκληθούν από πάρα πολλές αιτίες. Ο προγραμματιστής είναι υποχρεωμένος μέσα από το πρόγραμμα του να προβλέψει κάθε πιθανό λάθος που μπορεί να συμβεί στο περιβάλλον εκτέλεσης, ώστε να το αντιμετωπίσει και να αποφύγει δυσάρεστα αποτελέσματα για το χρήστη.

Κάθε λάθος παράγει έναν κωδικό λάθους. Συνήθως ο προγραμματιστής δημιουργεί γενικές διαδικασίες χειρισμού λαθών, που τις εντάσσει σε βιβλιοθήκες.

Η διαδικασία χειρισμού λαθών κατά το χρόνο εκτέλεσης είναι απλή. Η εκτέλεση κάθε εντολής προγράμματος επιστρέφει μια ένδειξη που περιγράφει, αν ολοκληρώθηκε επιτυχώς ή παρουσιάστηκε κάποιο λάθος. Όταν παρουσιαστεί ένα λάθος, ο προγραμματιστής πρέπει να εκμεταλλευτεί αυτή την ένδειξη και μέσα από εντολές κώδικα να δώσει δυνατότητες διαφυγής στο πρόγραμμα.

Πιο αναλυτικά ο χειρισμός ενός λάθους κατά το χρόνο εκτέλεσης περιγράφεται ως εξής:

Στην ενότητα που θέλουμε να προσθέσουμε δυνατότητες χειρισμού λαθών, χρησιμοποιούμε τον αντίστοιχο μηχανισμό ανίχνευσης λαθών που προσφέρει το περιβάλλον προγραμματισμού.

Όταν προκληθεί ένα λάθος, ο μηχανισμός ανίχνευσης μεταφέρει τη ροή εκτέλεσης του προγράμματος σε ένα τμήμα κώδικα χειρισμού λάθους (exception ή error handler), στο οποίο περιγράφει το λάθος τις περισσότερες φορές με κάποιο αριθμό. Το συγκεκριμένο τμήμα κώδικα συνήθως βρίσκεται μέσα στην ίδια ενότητα που προκαλείται το λάθος.

Για την ανίχνευση και το χειρισμό των λαθών που εμφανίζονται κατά την εκτέλεση ενός προγράμματος στις γλώσσες προγραμματισμού Java, ADA και C++ χρησιμοποιείται ο μηχανισμός των εξαιρέσεων (exceptions), ενώ στη Visual Basic υπάρχει ειδική εντολή (On Error GoTo).

Η ρουτίνα χειρισμού του λάθους είναι διαφορετική κάθε φορά και δημιουργείται από τον προγραμματιστή ειδικά για το χειρισμό των λαθών που μπορεί να προκύψουν από την εργασία που εκτελείται εκείνη τη χρονική στιγμή, με στόχο φυσικά την ανώδυνη απεμπλοκή του προγράμματος από την ενδεχόμενη προβληματική κατάσταση.

Στη συνέχεια περιγράφουμε ορισμένες ενέργειες που θα μπορούσαμε να συμπεριλάβουμε στο τμήμα κώδικα που θα γίνει ο χειρισμός του λάθους:

- να διορθώσουμε μέσα από εντολές κώδικα το λάθος ή να δώσουμε τη δυνατότητα στο χρήστη να διορθώσει την αιτία πρόκλησης του λάθους. Στη συνέχεια να επιστρέψουμε τη ροή εκτέλεσης στην εντολή που προκάλεσε αυτή την κατάσταση λάθους. Μια τέτοια κατάσταση μπορεί να προκληθεί από την προσπάθεια του χρήστη να εκτυπώσει κάποιο αρχείο, ενώ ο εκτυπωτής είναι κλειστός. Μόλις ανιχνευτεί το λάθος, ενημερώνουμε το χρήστη ότι ο εκτυπωτής είναι κλειστός και αφού βεβαιωθούμε ότι διορθώθηκε το λάθος, επιστρέφουμε τη ροή εκτέλεσης του προγράμματος στην εντολή εκτύπωσης, η οποία ολοκληρώνεται με επιτυχία.
- να αξιολογήσουμε την εντολή που προκάλεσε το σφάλμα και αν θεωρήσουμε ότι η εκτέλεσή της δεν είναι κρίσιμη για την εφαρμογή, να μεταφέρουμε τη ροή εκτέλεσης στην επόμενη εντολή. Η εκτέλεση ενός αρχείου ήχου ταυτόχρονα με την εκτέλεση μιας εργασίας (π.χ. αντιγραφή αρχείων) δεν είναι σημαντική, εφόσον η συγκεκριμένη εργασία ολοκληρώνεται με επιτυχία.

- να θεωρήσουμε ιδιαίτερα κρίσιμο το σφάλμα και αφού ενημερώσουμε το χρήστη, να τερματίσουμε το πρόγραμμα. Αυτός ο τρόπος φυσικά πρέπει να αποφεύγεται, γιατί οδηγεί την εκτέλεση του προγράμματος σε απότομη διακοπή με απρόβλεπτες συνέπειες.

Όλες οι παραπάνω εργασίες πραγματοποιούνται μέσα από κατάλληλες εντολές που προσφέρουν σχεδόν όλα τα σύγχρονα περιβάλλοντα προγραμματισμού. Υπάρχουν εντολές που ανιχνεύουν ένα λάθος, δίνουν την περιγραφή του (αριθμητικά ή και με μήνυμα) και επιτρέπουν τη μεταβίβαση της ροής εκτέλεσης του προγράμματος.

Ανακεφαλαίωση

Το ζητούμενο κάθε χρήστη είναι ένα φιλικό και σταθερό πρόγραμμα που θα εργάζεται κάτω από οποιεσδήποτε συνθήκες και θα προλαβαίνει όλες τις εσφαλμένες του επιλογές. Η σωστή εκσφαλμάτωση ενός προγράμματος έχει ως αποτέλεσμα την κατασκευή τέτοιων προγραμμάτων που θα παρουσιάσουν όσο το δυνατό λιγότερα προβλήματα στο χρήστη.

Η εκσφαλμάτωση αποτελεί μια από τις βασικές εργασίες που πρέπει να κάνει ο προγραμματιστής για την ολοκλήρωση ενός προγράμματος. Κατά τη διαδικασία της εκσφαλμάτωσης ο προγραμματιστής πρέπει να εντοπίσει και να διορθώσει τα σημεία του προγράμματος που τυχόν προκαλούν λάθη. Για την επίτευξη του σκοπού αυτού πρέπει να κάνει τους κατάλληλους ελέγχους και να αναλύσει το πρόγραμμά του μέσα από τα εργαλεία εκσφαλμάτωσης που του παρέχει το περιβάλλον προγραμματισμού.

Τέλος ο προγραμματιστής για να δημιουργήσει ασφαλή και ποιοτικά προγράμματα, πρέπει να προβλέψει όλες τις καταστάσεις λάθους που μπορεί να συμβούν στο περιβάλλον εκτέλεσης από κακό χειρισμό ή προβληματικό εξοπλισμό και να δώσει στο πρόγραμμά του δυνατότητες αντιμετώπισης και τρόπους διαφυγής.

Λέξεις κλειδιά

Λάθος, έλεγχος, εκσφαλμάτωση, εργαλεία εκσφαλμάτωσης, χειρισμός λάθους.

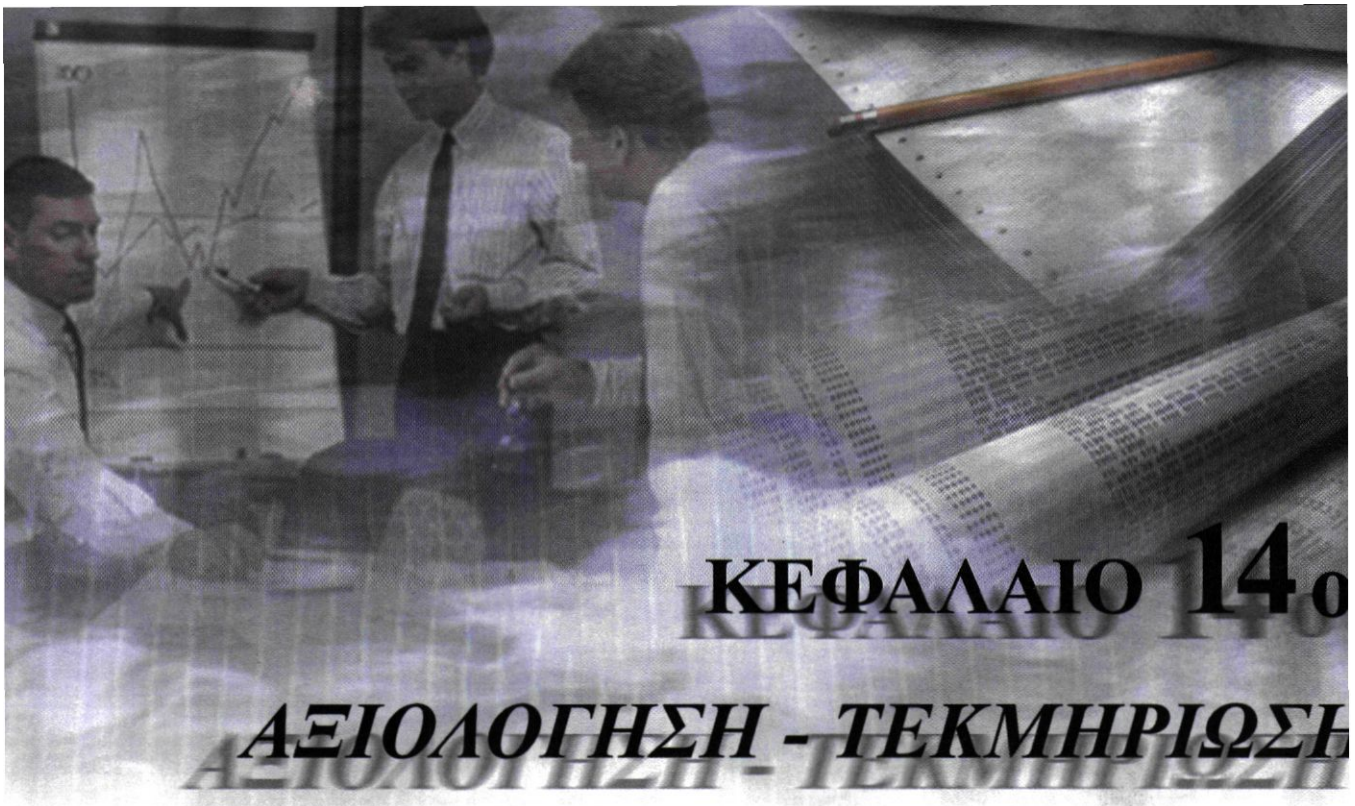
Ερωτήσεις - Θέματα για συζήτηση

1. Να δοθούν παραδείγματα λαθών που εμφανίζονται συχνά στο λογισμικό.

2. Να γίνει συζήτηση σχετικά με τις επιπτώσεις των λαθών του λογισμικού.
3. Περιγράψτε το χειρισμό ενός λάθους που εμφανίζεται κατά το χρόνο εκτέλεσης ενός προγράμματος.
4. Περιγράψτε τη χρησιμότητα των εργαλείων εκσφαλμάτωσης.
5. Να γίνει συζήτηση για τις αιτίες που οδηγούν στην κατασκευή προβληματικού λογισμικού.

Βιβλιογραφία

1. Εγκυκλοπαίδεια Πληροφορικής και Τεχνολογίας Υπολογιστών, Εκδόσεις Νέων Τεχνολογιών, Τόμος 2, Αθήνα, 1986.
2. Παν. Πολίτης-Ηλ. Γιαννόπουλος: Προγραμματισμός με τη Visual Basic 4.0, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1997.
3. Ivars Peterson: Fatal Defect:Chasing killer Computer Bugs, Science News, USA, 1995.



ΚΕΦΑΛΑΙΟ 14ο

ΑΞΙΟΛΟΓΗΣΗ - ΤΕΚΜΗΡΙΩΣΗ

ΠΕΡΙΕΧΟΜΕΝΑ

- Κριτήρια αξιολόγησης
- Τεκμηρίωση προγράμματος
- Κύκλος ζωής λογισμικού

Εισαγωγή

Η αξιολόγηση είναι το τελευταίο στάδιο στην παραγωγή ενός προγράμματος. Σ' αυτό το κεφάλαιο θα αναφέρουμε τα κριτήρια βάση των οποίων αξιολογείται ένα πρόγραμμα, θα δώσουμε διάφορα παραδείγματα και θα απαντήσουμε ερωτήσεις όπως "γιατί αυτή η λύση είναι προτιμότερη από την άλλη", στηρίζοντας την απόφασή μας σε αυτά τα κριτήρια. Θα συζητήσουμε την αναγκαιότητα, τη σημασία, το σκοπό και τις κατηγορίες της τεκμηρίωσης, καθώς και τη σημασία της κατά φάση συντήρησης του προγράμματος. Θα δώσουμε οδηγίες για την σύνταξη ενός VTOC (Visual Table Of Contains), μιας διαγραμματικής τεχνικής που χρησιμοποιείται στην τεκμηρίωση ενός προγράμματος. Θα συζητήσουμε για τις φάσεις ανάπτυξης ενός προγράμματος και θα δούμε πως σχετίζονται με τις φάσεις του κύκλου ζωής ενός προγράμματος.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να διατυπώνουν για το ίδιο πρόβλημα εναλλακτικές προγραμματιστικές λύσεις, να τις συγκρίνουν και να τις αξιολογούν με βάση προκαθορισμένα κριτήρια.
- να προσδιορίζουν τα όρια χρήσης κάθε προγράμματος που δημιουργούν.
- να αναζητούν και να διερευνούν τις δυνατότητες επέκτασης των προγραμμάτων που δημιουργούν (νέες πρόσθετες λειτουργίες, κ.λπ.).
- να αξιολογούν τα προγράμματα που δημιουργούν.
- να διακρίνουν ποιες είναι οι αναγκαίες σημειώσεις για την τεκμηρίωση του προγράμματος.
- να συντάσσουν το φάκελο τεκμηρίωσης της εφαρμογής.
- να διακρίνουν τα στάδια του κύκλου ζωής ενός προγράμματος

Προερωτήσεις

- ✓ Πιστεύεις ότι το τέλος της δημιουργίας ενός προγράμματος, είναι το να το θέσεις σε παραγωγή;
- ✓ Νομίζεις ότι η λύση του προβλήματος που διατύπωσες είναι μοναδική;
- ✓ Γνωρίζεις τι είναι πακέτο λογισμικού και ποια η διαφορά του με το πρόγραμμα;
- ✓ 'Εχεις ακούσει για τον φάκελο τεκμηρίωσης προγράμματος;

14.1 Κριτήρια αξιολόγησης προγράμματος

Οι προτεινόμενες λύσεις των προβλημάτων του κεφαλαίου, ακολουθούν σε μεγάλο βαθμό τα κριτήρια αξιολόγησης προγράμματος, προσπαθώντας να κρατήσουν μια ισορροπία μεταξύ τους.

Ένα πρόγραμμα αξιολογείται όχι μόνο από το αν λειτουργεί ή όχι, πράγμα αυτονόητο, αλλά και από το αν πληρεί κάποια κριτήρια.

Ένα πρόγραμμα όπως όλα τα προϊόντα αξιολογείται και από τον κατασκευαστή του αλλά και από το χρήστη του. Επομένως τα κριτήρια αξιολόγησης πρέπει να καλύπτουν και τις δύο πλευρές. Έτσι ένα ποιοτικό λογισμικό πρέπει να έχει:

- ✓ Απλότητα – Τυπικότητα (απλές δομές, ίδια αντιμετώπιση ίδιων προβλημάτων)
- ✓ Φιλικότητα (ανοχή στα λάθη, πληροφόρηση και πολλά άλλα)
- ✓ Ευελιξία (δυνατότητα επεκτάσεων και τροποποιήσεων)
- ✓ Αξιοπιστία (εκτέλεση χωρίς λάθη)
- ✓ Ταχύτητα

Τα κριτήρια της απλότητας και τυπικότητας, της ευελιξίας, της αξιοπιστίας και της ταχύτητας, αφορούν τεχνικά θέματα και είναι τα κριτήρια που χρησιμοποιεί ο προγραμματιστής, ο κατασκευαστής του προγράμματος, προκειμένου να αξιολογήσει το έργο του. Το κριτήριο της φιλικότητας είναι το κατ' εξοχήν κριτήριο που αξιολογείται από το χρήστη της εφαρμογής. Το τελευταίο λόγω της πολυπλοκότητας του έχει αναπτυχθεί διεξοδικά σε ξεχωριστό κεφάλαιο. Εδώ θα αναπτύξουμε και θα δώσουμε παραδείγματα για τα υπόλοιπα.

Αρκετές φορές προσπαθώντας να τηρήσει κανείς τα κριτήρια αυτά, βρίσκεται μπροστά σε διλήμματα. Πράγματι στην προσπάθεια να ικανοποιηθεί κάποιο από αυτά, μπορεί το πρόγραμμά να υστερήσει σε κάποιο άλλο. Σ' αυτές τις περιπτώσεις, η πείρα σε συνδυασμό με τη σπουδαιότητα του συγκεκριμένου χαρακτηριστικού, θα δώσουν την απάντηση στο δίλημμα.

14.1.1 Απλότητα - τυπικότητα

Στα χρόνια που ο σχεδιασμός ενός προγράμματος ήταν τέχνη, κάθε αλγόριθμος έφερνε την "υπογραφή" του κατασκευαστή του. Συνήθεια της εποχής ήταν η υιοθέτηση περίπλοκων τεχνικών (κάτι σαν πνευματικές ασκήσεις), που άλλοτε είχαν στόχο να λύσουν υπαρκτά προβλήματα, άλλοτε είχαν στόχο το πρόγραμμα να είναι ακατανόητο για οποιονδήποτε τρίτο και άλλοτε να δείξουν την επιδεξιότητα του προγραμματιστή. Η πορεία έδειξε ότι οι λύσεις που παράγονταν με αυτό τον τρόπο, δεν ήταν οι καλύτε-

ρες. Οι ανάγκες συντήρησης των προγραμμάτων, αλλά και η εξέλιξη του υλικού, έφεραν σταδιακά αυτήν την τακτική στο περιθώριο. Στο προσκήνιο βρέθηκε η απλότητα. Σήμερα πλέον στα προγράμματα που κατασκευάζονται:

Το απλό είναι ωραίο !!

✓ Εύκολα γίνεται αντιληπτό τι λειτουργίες εκτελούν.

✓ Εύκολα μπορούν να συμπληρωθούν από άλλους.

Παρ όλα αυτά και σήμερα προικισμένοι προγραμματιστές, ολισθαίνουν προς το σχεδιασμό ενός περίπλοκου προγράμματος, χρησιμοποιώντας συνήθως "ισχυρότερες" δομές από αυτές που είναι αναγκαίες.

Με το επόμενο παράδειγμα δίνονται για το ίδιο πρόβλημα τρεις διαφορετικές λύσεις και σχολιάζονται κατά πόσον ικανοποιούν το κριτήριο της απλότητας και της τυπικότητας.

Παράδειγμα 1

Δίδεται ταξινομημένος μονοδιάστατος πίνακας ακεραίων αριθμών 1000 θέσεων. Ζητείται να κατασκευαστεί πρόγραμμα, το οποίο να βρίσκει τη συχνότητα εμφάνισης κάθε αριθμού του πίνακα. Τα αποτελέσματα να εμφανίζονται στην οθόνη.

Ένας απλός και τυπικός αλγόριθμος θα πρέπει:

- Να δίνει τις ίδιες τιμές στις μεταβλητές, είτε είναι η αρχή του προγράμματος, είτε πρόκειται για νέο αριθμό.
- Αυτό που γίνεται σπανιότερα (<>), εδώ η αλλαγή του αριθμού, να ξεχωρίζει μέσα σε ένα **AN**, ενώ αυτό που γίνεται συνήθως (=) να βρίσκεται στην κανονική ροή του προγράμματος.
- Ένα **AN** μπορεί να πραγματοποιεί τον επιθυμητό έλεγχο.

Έστω ότι τα περιεχόμενα του πίνακα είναι 2, 2, 3, 3, 3, 3, 5, 6, 6, 6, 6, 6, 7, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9. Τότε το πρόγραμμα θα πρέπει να εμφανίσει τα εξής αποτελέσματα:

Αριθμός	Συχνότητα
2	2
3	4
5	1
6	5
7	1
8	6
9	6

Τα επόμενα τρία προγράμματα λύνουν το πρόβλημα.

```
ΠΡΟΓΡΑΜΜΑ Συχνότητα1
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, S, Προηγ_Α, Α[1000]
ΑΡΧΗ
i <- 1
S <- 0
Προηγ_Α <- Α[i]
ΟΣΟ i < 1001 ΕΠΑΝΑΛΑΒΕ
    ΑΝ Προηγ_Α <> Α[i] ΤΟΤΕ
            ΓΡΑΨΕ Προηγ_Α, S
            Προηγ_Α <- Α[i]
            S <- 0
    ΤΕΛΟΣ_ΑΝ
    S <- S+1
    i <- i+1
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ Προηγ_Α, S
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Συχνότητα1

ΠΡΟΓΡΑΜΜΑ Συχνότητα2
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, S, Προηγ_Α, Α[1000]
ΑΡΧΗ
i <- 1
S <- 0
Προηγ_Α <- Α[i]
ΟΣΟ i < 1001 ΕΠΑΝΑΛΑΒΕ
    ΟΣΟ Προηγ_Α = Α[i] ΚΑΙ i < 1001 ΕΠΑΝΑΛΑΒΕ
        S <- S+1
```

```
        i <- i+1
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ Προηγ_Α, S
    Προηγ.Α <- Α(i)
    S <- 0
ΤΕΛΟΣ_ ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Συχνότητα2

ΠΡΟΓΡΑΜΜΑ Συχνότητα3
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: i, S, Προηγ_Α, Α[1000]
ΑΡΧΗ
i <- 1
S <- 0
Προηγ_Α <- Α[i]
ΟΣΟ i < 1001 ΕΠΑΝΑΛΑΒΕ
    ΑΝ Προηγ_Α= Α[i] ΤΟΤΕ
        S <- S+1
        i <- i+1
    ΑΛΛΙΩΣ
        ΓΡΑΨΕ Προηγ_ Α, S
        Προηγ_Α <- Α[i]
        S <- 0
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ Προηγ_Α, S
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Συχνότητα3
```

Αρχικά παρατηρούμε ότι όλες οι λύσεις πληρούν όλους τους κανόνες του δομημένου προγραμματισμού και γενικότερα όλων των αρχών και κανόνων που θέσαμε σ' αυτό το βιβλίο.

Στην πρώτη λύση συναντούμε μία λύση με αδρές γραμμές. Πριν την επανάληψη τοποθετούνται αρχικές τιμές και δίνεται η πρώτη τιμή στο Α(i). Μέσα στο βρόχο ξεχωρίζει το τμήμα που είναι για όλα τα Α(i), και το τμήμα που εκτελείται, όταν διαπιστωθεί διαφορετικό Α(i). Τέλος μετά την επανάληψη έχουμε ένα μέρος των εντολών που εκτελούνται και όταν βρίσκεται διαφορετικό Α(i).

Στη δεύτερη λύση για κάποιες τιμές του i θα γίνει δύο φορές η ερώτηση i<1001. Αυτό θα συμβεί όταν βρεθεί το Προηγ_Α(i). Τότε η εσωτερική επαναληπτική διαδικασία θα τερματιστεί, θα γίνουν η εκτύπωση κ.λπ. και αμέσως μετά θα γίνει η ερώτηση i<1001, χωρίς στο μεταξύ το i να έχει αυξηθεί.

'Ενας λόγος που συνέβη αυτό είναι ότι, χρησιμοποιήθηκε ή δομή **ΟΣΟ** ... **ΕΠΑΝΑΛΑΒΕ** που είναι ισχυρότερη της **ΑΝ** που πραγματικά χρειάζεται.

Στην τρίτη λύση δεν έχουμε τυποποίηση, γιατί η λύση δίνει την ίδια βαρύτητα στο συνηθισμένο γεγονός (=) και στο σπάνιο (<>), τοποθετώντας τα μέσα σε ένα **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ**.

Συμπερασματικά:

✓ Η πρώτη λύση είναι απλούστερη και τυπικότερη.

✓ Η δεύτερη λύση δεν πληρεί το κριτήριο της απλότητας, μια που χρησιμοποιεί δομή περισσότερο σύνθετη απ' όσο χρειάζεται.

✓ Η τρίτη λύση είναι απλή, υστερεί όμως σε τυπικότητα.

Παρατηρήστε ότι και στις τρεις λύσεις, οι ενέργειες που γίνονται όταν αλλάζει το περιεχόμενο του A(i) χωρίζονται σε δύο επιμέρους τμήματα. Το ένα "ΓΡΑΨΕ Προηγ.A, S" επαναλαμβάνεται το ίδιο πριν το τέλος του προγράμματος, ενώ το άλλο "Προηγ.A<-A(i), s<-0" έχει γραφτεί για πρώτη φορά πριν την αρχή της επαναληπτικής διαδικασίας. Ο χωρισμός γίνεται γιατί, στο κάθε τμήμα μπορούμε να διακρίνουμε συγκεκριμένο σκοπό. Το τμήμα με την εντολή "ΓΡΑΨΕ Προηγ.A, S" είναι η βασική επεξεργασία, η καρδιά του προγράμματος, εκτελείται κάθε φορά που αλλάζει ο αριθμός, αλλά και μετά το τέλος της επαναληπτικής διαδικασίας. Το άλλο τμήμα, με τις εντολές "s<-0, Προηγ.A<-A(i)" έχει σκοπό να δώσει τις αρχικές τιμές στις μεταβλητές και φυσικά εκτελείται πριν ακόμα αρχίσει η επαναληπτική διαδικασία αλλά και μέσα σε αυτήν. Σ' αυτά τα προγράμματα υπάρχει συνήθως, μία ακόμα ενότητα που είναι η προετοιμασία της βασικής επεξεργασίας. Εδώ αυτή αποτελείται από μία μόνο εντολή την s<-s+1.

14.1.2 Ευελιξία

'Ενα από τα κριτήρια αξιολόγησης του προγράμματος είναι και η δυνατότητα επέκτασης του. Παρατηρώντας τον κύκλο ζωής προγράμματος, που αναφέρεται παρακάτω, παρατηρούμε ότι η μισή ζωή του προγράμματος θα καταναλωθεί για τις βελτιώσεις, προσθήκες, γενικά για τις αλλαγές στο αρχικό πρόγραμμα. Αυτό δίνει μεγαλύτερη βαρύτητα στο κριτήριο της ευελιξίας. Ο σχεδιασμός ενός ευέλικτου προγράμματος απαιτεί περισσότερο χρόνο για την επιλογή της μεθόδου, που θα ακολουθήσουμε για τη λύση του.

'Ενα κλασικό παράδειγμα, όπου ο αρχικός σχεδιασμός δεν επιτρέπει την επέκταση του αλγόριθμου παρ' ότι είναι λογικά σωστός, είναι η σύγκριση 3 αριθμών και η κατάταξή τους σε σειρά. Λογική επέκταση του θα ήταν ένα

πρόβλημα που ζητά την κατάταξη 4 αριθμών. Αν η μετάβαση από τον ένα αλγόριθμο στον άλλο είναι γρήγορη, τότε έχουμε σχεδιάσει έναν αλγόριθμο σε σωστή βάση.

Ας δούμε την λογική του εξέλιξη, δεδομένου ότι το πρόβλημα είναι εξαντλημένο από μαθηματικής πλευράς.

Παράδειγμα 2

Δίνεται ζεύγος αριθμών από το πληκτρολόγιο και ζητείται να εμφανίζονται με αύξουσα σειρά στην οθόνη.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_2α
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β
ΔΙΑΒΑΣΕ α, β
ΑΝ α<β ΤΟΤΕ
    ΓΡΑΨΕ α, β
ΑΛΛΙΩΣ
    ΓΡΑΨΕ β, α
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_2α
```

Το παραπάνω πρόγραμμα είναι φυσικά σωστό, απλό, και πληρεί όλους τους όρους που έχουν τεθεί.

Είναι όμως επεκτάσιμο;

Ας δούμε την πιθανότερη επέκταση του.

Παράδειγμα 3

Δίνεται τριάδα αριθμών από το πληκτρολόγιο και ζητείται να εμφανιστούν με αύξουσα σειρά στην οθόνη.

Οι διαφορετικές διατάξεις 3 αριθμών είναι 6 (3!=1*2*3=6). έτσι η παρακάτω λύση για τρεις αριθμούς, είναι η λογική επέκταση της προηγουμένης.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_3α
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β, γ
ΔΙΑΒΑΣΕ α, β, γ
ΑΝ α<β ΤΟΤΕ
    ΑΝ β<γ ΤΟΤΕ
        ΓΡΑΨΕ α, β, γ
```

```
    ΑΛΛΙΩΣ
        ΑΝ α<γ ΤΟΤΕ
            ΓΡΑΨΕ α, γ, β
        ΑΛΛΙΩΣ
            ΓΡΑΨΕ γ, α, β
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ ΑΝ
ΑΛΛΙΩΣ
    ΑΝ α<γ ΤΟΤΕ
        ΓΡΑΨΕ β, α, γ
    ΑΛΛΙΩΣ
        ΑΝ β<γ ΤΟΤΕ
            ΓΡΑΨΕ β, γ, α
        ΑΛΛΙΩΣ
            ΓΡΑΨΕ γ, β, α
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_3α
```

Αν τώρα μας ζητηθεί η ταξινόμηση τεσσάρων αριθμών, θα πρέπει πρώτα να βρούμε τις 24 (4!=1*2*3*4=24) διαφορετικές διατάξεις των τεσσάρων αριθμών και να γράψουμε φυσικά τα ανάλογα **ΑΝ,** προκειμένου να αναγνωρίσει το πρόγραμμα για ποια διάταξη πρόκειται, ώστε να την εμφανίσει στην οθόνη.

Παρατηρούμε ότι ενώ έχουμε λύσει το πρώτο πρόβλημα με απλό τρόπο, εν τούτοις όταν τα δεδομένα στην είσοδο αυξάνουν, γίνεται όλο και περισσότερο επίπονο να κατασκευαστεί αλγόριθμος που να καλύπτει την νέα ανάγκη, παρ' όλο ότι αυτή δεν φαίνεται αρκετή να προκαλέσει τέτοια αναστάτωση.

Ας δούμε λοιπόν τι πήγε λάθος, ώστε ένα πρόγραμμα που ο βασικός του κορμός δείχνει να έχει σχεδιαστεί σωστά, δεν αντέχει αυτό που από πολλούς θεωρείται φυσική του επέκταση.

Το λάθος προήλθε από το ότι στηριχθήκαμε για τη λύση του, σε μία λύση γνωστή από τα μαθηματικά, που είναι πολύ κοντά στην ανθρώπινη αντιμετώπιση του προβλήματος. Θεωρήσαμε ότι αυτή είναι "καλή λύση" και προχωρήσαμε και στην πρώτη επέκταση της, χωρίς να υπολογίσουμε ότι στον κύκλο ζωής ενός προγράμματος, ο αρχικός σχεδιασμός είναι μόνο το 16%, ενώ η συντήρηση, που περιλαμβάνει και τις επεκτάσεις, είναι 50%.

Ας δούμε μία λύση που στοχεύει στο μεγαλύτερο ποσοστό. Μια λύση που θα είναι εύκολα επεκτάσιμη.

Συμπερασματικά λοιπόν μπορούμε να πούμε ότι λύσεις που είναι καλές για μας μπορεί να μην είναι οι κατάλληλες για τον υπολογιστή. Αυτό θα πρέπει να εξετάζεται σαν μία σοβαρή παράμετρος στην επιλογή του αλγόριθμου.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_2β
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β
ΔΙΑΒΑΣΕ α, β
ΑΝ α>β ΤΟΤΕ
    temp <- α
    α<-β
    β<-temp
ΤΕΛΟΣ_ΑΝ
ΓΡΑΨΕ α, β
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_2β
```

Στο παραπάνω πρόγραμμα τυποποιήθηκε η έξοδος, ώστε πάντα να τυπώνεται το περιεχόμενο του α και μετά του β. Αυτό προς στιγμήν αυτή η τυποποίηση της εξόδου, δεν δείχνει την σημασία της, αντίθετα θα έλεγε κανείς ότι η λύση είναι "περίεργη" και θα είχε δίκιο.

Τώρα ας επεκτείνουμε αυτήν τη λογική για να λύσουμε το πρόβλημα για τρεις αριθμούς.

```
ΠΡΟΓΡΑΜΜΑ Σειρά_3β
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: σ
    ΠΡΑΓΜΑΤΙΚΕΣ α, β, γ
σ <- 0
ΔΙΑΒΑΣΕ α, β, γ
ΟΣΟ σ=0 ΕΠΑΝΑΛΑΒΕ
    σ <- 1
    ΑΝ α>β ΤΟΤΕ
        temp <- α
        α <- β
        β <- temp
        σ <- 0
    ΤΕΛΟΣ_ΑΝ
    ΑΝ β>γ ΤΟΤΕ
        temp <- β
        β <- γ
        γ <- temp
        σ <- 0
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ α, β, γ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ Σειρά_3β
```

Η μεταβλητή σ δείχνει, αν έχει γίνει αντιμετάθεση στα ζεύγη που συγκρίνονται. Αν δεν έχει γίνει καμία τότε σ=1 και οι αριθμοί είναι στη σωστή σειρά και άρα μπορούν να εμφανιστούν. Εξετάζοντας το προηγούμενο πρόγραμμα παρατηρούμε ότι, υπάρχει μια σειριακή ακολουθία από ανεξάρτητα **AN** (μπορούν να αλλάξουν σειρά), στα οποία συγκρίνονται οι αριθμοί με τη σειρά εισαγωγής τους. Έτσι γίνονται οι συγκρίσεις α>β και β>γ και γίνεται η αντιμετάθεσή τους, αν χρειάζεται. Δηλαδή από τη σύγκριση των δύο (α>β) περάσαμε στην σύγκριση των τριών (α>β, και β>γ).

Τώρα η σύγκριση δύο αριθμών με τη χρήση αντιμετάθεσης δεν φαίνεται πια "περίεργη".

Στα παραπάνω είδαμε τα βήματα **βελτιστοποίησης** της δομής ενός προγράμματος με την ευκαιρία μιας επέκτασης που δεν μπορούσε να γίνει.

14.1.3 Αξιοπιστία

Ο όρος αναφέρεται στη συνεχή και ομοιόμορφη λειτουργία ενός προγράμματος. Πράγματι χρησιμοποιώντας ένα πρόγραμμα περιμένουμε από αυτό, να συμπεριφέρεται βάση των προδιαγραφών, ακόμα και κάτω από τις δυσμενέστερες συνθήκες.

Συνήθως τα προβλήματα αξιοπιστίας παρουσιάζονται σε πραγματικές συνθήκες λειτουργίας των προγραμμάτων, όταν ο χρήστης του προγράμματος δεν είναι πλέον ο κ. Τέλειος αλλά αντίθετα ο κ. Άσχετος.

Συνήθη λάθη χειρισμού, ή ακόμη ηθελημένα λάθη που επιζητούν τα όρια του προγράμματος, μπορούν να καταρρακώσουν την αξιοπιστία του.

- Η αξιοπιστία ενός προγράμματος αυξάνεται με επιτυχείς ελέγχους επί των δεδομένων.
- Η αξιοπιστία ενός προγράμματος μειώνεται, όταν αποδέχεται δεδομένα εκτός προδιαγραφών και προχωρά στην επεξεργασία τους.
- Σε ένα αξιόπιστο πρόγραμμα δεν πρέπει να υπάρχουν εντολές ή ενότητες που δεν εκτελούνται.
- Ένα πρόγραμμα για να παραμένει αξιόπιστο, θα πρέπει μετά από κάθε αλλαγή, να επανελέγχεται με όλα τα δεδομένα ελέγχου που είχαν χρησιμοποιηθεί στους προηγούμενους ελέγχους.
- Σε ένα αξιόπιστο πρόγραμμα πρέπει να αντιμετωπίζονται όλες οι ακραίες περιπτώσεις, δεδομένων και ενεργειών.

Ας δώσουμε ένα παράδειγμα. Ας υποθέσουμε ότι έχουμε ένα πρόγραμμα που διαβάζει δύο αριθμούς και εμφανίζει το άθροισμα τους. Προφανώς από το πρόγραμμα περιμένουμε, όσο του δίνουμε αριθμούς, να μας δίνει το άθροισμα. Τι θα κάνει όμως το πρόγραμμα, αν του δώσουμε ετερόσημους αριθμούς; θα πρέπει να μας δώσει το αλγεβρικό τους άθροισμα. Το ίδιο θα

πρέπει να συμβεί και αν του δώσουμε δύο αρνητικούς αριθμούς. Αν του δώσουμε έναν αριθμό και ένα γράμμα όμως; Εδώ έχουμε αλλαγή στον τύπο δεδομένων και ένα πρόγραμμα που δουλεύει αξιόπιστα, πρέπει να απορρίψει το γράμμα σαν δεδομένο ακατάλληλο για επεξεργασία και να ζητήσει άλλο αποδεκτό τύπο δεδομένου.

Σε άλλη περίπτωση δίνεται ημερομηνία από το πληκτρολόγιο (ημέρα, μήνας, έτος), και ζητείται να βρεθεί σε ποια ημέρα της εβδομάδας αντιστοιχεί. 'Αραγε πως θα συμπεριφερθεί το πρόγραμμα αν του δώσουμε ενδεικτικά: Αρνητικό αριθμό ή μηδέν στην θέση της ημέρας, του μήνα ή του έτους; Αριθμό μήνα μεγαλύτερο του 12; Αριθμό ημέρας 32; Εδώ έχουμε τιμές εκτός ορίων υπολογισμού, προφανώς ένα αξιόπιστο πρόγραμμα πρέπει να απορρίπτει κάθε τέτοια ημερομηνία και να ζητά άλλη αποδεκτή τιμή ημερομηνίας.

Παραδείγματα τιμών άλλου τύπου από τον αναμενόμενο είναι και τα:

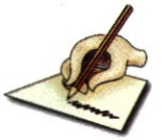
- ✓ Τι πρέπει να κάνει ένα πρόγραμμα που στα γράμματα ενός ονοματεπώνυμου δίνονται και αριθμοί;
- ✓ Τι πρέπει να κάνει ένα πρόγραμμα όταν τα ψηφία του ταχυδρομικού κώδικα είναι λιγότερα από πέντε;
- ✓ Τι πρέπει να κάνει ένα πρόγραμμα που του δίδονται γράμματα στη θέση αριθμών;

Παραδείγματα τιμών εκτός ορίου υπολογισμών είναι και τα:

- Δίνεται δυαδικός αριθμός. Ζητείται να μετατραπεί στον αντίστοιχό του δεκαδικό. Προφανώς στην είσοδο πρέπει να γίνονται δεκτά μόνο ψηφία 0 και 1.
- Δίνεται ρωμαϊκός αριθμός. Ζητείται να μετατραπεί στον αντίστοιχό του δεκαδικό. Προφανώς στην είσοδο πρέπει κατ' αρχήν να γίνονται δεκτοί μόνο χαρακτήρες I, V, X, L, C, D, M, αλλά και μία ακόμα σειρά ελέγχων προκειμένου να γίνει αποδεκτός ο συνδυασμός των ορθών χαρακτήρων.

Για τις περιπτώσεις που η αξιοπιστία του προγράμματος κινδυνεύει από διαφορετικό τύπο δεδομένου, ο τρόπος ελέγχου προσφέρεται από τη γλώσσα προγραμματισμού. Πράγματι κάθε γλώσσα έχει τη δυνατότητα να ορίσει τύπο μεταβλητής και έτσι να αποκλείσει κάθε άλλο τύπο δεδομένων μέσα στη μεταβλητή αυτή. Αυτά τα προβλήματα είναι απλούστερο να αντιμετωπιστούν, δεδομένου ότι ο έλεγχος του τύπου πραγματοποιείται αυτόματα από τη γλώσσα προγραμματισμού.

Για την περίπτωση των τιμών εκτός ορίου, ο τρόπος βελτίωσης της αξιοπιστίας θα στηριχθεί στους ελέγχους που θα συμπεριληφθούν στο πρόγραμμα. Προσπαθώντας για τη λύση τέτοιου είδους προβλημάτων, διαπιστώνεται ότι πρώτα πρέπει να σχεδιαστούν όλοι οι έλεγχοι και μετά η λύση του προβλήματος. Υπάρχει δηλαδή ένα *"πρόβλημα μέσα στο πρόβλημα"*.



Χωρίς πολύ καλή γνώση του χώρου του βασικού προβλήματος είναι αδύνατον να σχεδιαστούν σωστά οι απαραίτητοι έλεγχοι, ώστε το πρόγραμμα να είναι αξιόπιστο.

Διαπιστώνεται ακόμη, ότι ένα πρόγραμμα λίγων γραμμών, λόγω των ελέγχων, κύρια των ελέγχων αποδεκτών τιμών διογκώνεται υπέρμετρα.

Δεν θα αναφερθούμε σε παραδείγματα της πρώτης περίπτωσης, επειδή η αντιμετώπιση τους είναι στενά συνδεδεμένη με τη χρησιμοποιούμενη γλώσσα προγραμματισμού.

Ας δούμε λοιπόν ένα παράδειγμα για τη δεύτερη περίπτωση.

Παράδειγμα 4

Να κατασκευαστεί πρόγραμμα που θα ελέγχει μια ημερομηνία που δίνεται από το πληκτρολόγιο.

Κατ΄ αρχήν μερικές γνωστές και "άγνωστες" πληροφορίες για τον χώρο των ημερομηνιών, προκειμένου να υπάρχει ένα ελάχιστο ποσό γνώσεων για να περιγραφούν οι σχετικοί έλεγχοι.

- Κάθε έτος έχει 365 ημέρες, εκτός αν είναι δίσεκτο, οπότε έχει 366.
- Οι μήνες Ιανουάριος, Μάρτιος, Μάιος, Ιούλιος, Αύγουστος, Οκτώβριος και Δεκέμβριος έχουν 31 μέρες.
- Οι μήνες Απρίλιος, Ιούνιος, Σεπτέμβριος και Νοέμβριος, έχουν 30 μέρες.
- Ο Φεβρουάριος έχει 28 μέρες, εκτός αν το έτος είναι δίσεκτο, οπότε έχει 29.
- Ένα έτος λέγεται δίσεκτο αν ο αριθμός του είναι πολλαπλάσιο του 4, αλλά όχι του 100, εκτός αν είναι πολλαπλάσιο του 400. Για παράδειγμα: το 1984 είναι δίσεκτο (πολλαπλάσιο του 4), το 1900 δεν είναι (πολλαπλάσιο του 4 αλλά και του 100), το 2000 είναι δίσεκτο (πολλαπλάσιο του 4, του 100, αλλά και του 400), τέλος το 1993 δεν είναι.
- Το ημερολόγιο που ισχύει σήμερα είναι το Γρηγοριανό. Το εφάρμοσε ο πάπας Γρηγόριος ο ΙΓ' την 4η Οκτωβρίου του 1582 προσθέτοντας δέκα ημέρες. Έτσι η επόμενη της 4ης Οκτωβρίου ήταν η 15η Οκτωβρίου. Στην Ελλάδα εφαρμόστηκε στις 10 Μαρτίου 1923. Έτσι η επόμενη της 10ης Μαρτίου ήταν η 23η Μαρτίου 1923.

Προσέξτε λοιπόν γιατί δεν υπάρχουν στα ελληνικά ημερολόγια οι ημερομηνίες 11/3/1923 έως 22/3/1923. Ενώ στα Ευρωπαϊκά δεν υπάρχουν οι ημερομηνίες 5/10/1582 έως 14/10/1582. Αυτό σε κάποιες περιπτώσεις μπορεί να είναι σημαντικό, ενώ για άλλες αδιάφορο.

Μετά από αυτά σκεπτόμενοι τη λύση του προβλήματος, διαπιστώνουμε ότι θα πρέπει να κάνουμε τους παρακάτω ελέγχους.

- ✓ Πρώτα να ελέγξουμε, αν δόθηκαν αρνητικοί αριθμοί στην ημερομηνία.
- ✓ Μετά να ελέγξουμε, αν η ημέρα είναι υπαρκτή. Βλέπε πληροφορία για το Γρηγοριανό ημερολόγιο.
- ✓ Στη συνέχεια να ελέγξουμε, αν το έτος που δόθηκε είναι δίσεκτο. Αυτή την πληροφορία θα τη χρησιμοποιήσουμε στον έλεγχο της ημέρας, όταν ο μήνας είναι ο Φεβρουάριος.
- ✓ Και τέλος να ελέγξουμε, αν συνδυάζονται σωστά ο αριθμός της ημέρας με τον αριθμό του μήνα.

```
ΠΡΟΓΡΑΜΜΑ Ημερομηνία
ΔΙΑΒΑΣΕ η, μ, ε
ΚΑΛΕΣΕ Ελεγχος_1(η, μ, ε, c)
ΑΝ c=0 ΤΟΤΕ ΓΡΑΨΕ 'Αρνητικές τιμές'
ΚΑΛΕΣΕ Ελεγχος_2(η, μ, ε, c)
ΑΝ c=0 ΤΟΤΕ ΓΡΑΨΕ 'Ανύπαρκτη ημερομηνία'
ΚΑΛΕΣΕ Ελεγχος_3(η, μ, ε, c)
ΑΝ c=0 ΤΟΤΕ ΓΡΑΨΕ 'Λανθασμένη ημερομηνία'
ΤΕΛΟΣ

ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_1(η, μ, ε, c)
! Αρνητικές ημερομηνίες
c <- 0
ΑΝ ε>0 ΚΑΙ μ>0 ΚΑΙ η>0 ΤΟΤΕ c <- 1
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_1

ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_2(η, μ, ε, c)
! Ιουλιανό ημερολόγιο
c <- 1
ΑΝ (10<η ΚΑΙ η<23) ΚΑΙ μ=3 ΚΑΙ ε=1923 ΤΟΤΕ c <- 0
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_2

ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_3(η, μ, ε, c)
! 'Ελεγχος δίσεκτου έτους (αν δ=1 ΔΙΣΕΚΤΟ)
ε4 <- ε mod 4
ε100 <- ε mod 100
```

```
ε400 <- ε mod 400
δ <- 0
ΑΝ  ε4=0  ΤΟΤΕ
    ΑΝ  ε100=0 ΤΟΤΕ
        ΑΝ  ε400=0  ΤΟΤΕ
            δ <- 1
        ΤΕΛΟΣ_ΑΝ
    ΑΛΛΙΩΣ
        δ <- 1
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
! Ελεγχος αποδεκτής ημερομηνίας (c=1 ΑΠΟΔΕΚΤΗ)
c <- 0
ΕΠΙΛΕΞΕ μ
    ΠΕΡΙΠΤΩΣΗ 1, 3, 5, 7, 8, 10, 12
        ΑΝ η<=31 ΤΟΤΕ c <- 1
    ΠΕΡΙΠΤΩΣΗ 4, 6, 9, 11
        ΑΝ η<=30 ΤΟΤΕ c <- 1
    ΠΕΡΙΠΤΩΣΗ 2
        ΑΝ δ=0 ΚΑΙ η<=28 ΤΟΤΕ c <- 1
        ΑΝ δ=1 ΚΑΙ η<=29 ΤΟΤΕ c <- 1
ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
ΤΕΛΟΣ_ΔΙΑΔΙΚΑΣΙΑ Ελεγχος_3
```

Ας δοθεί προσοχή στη σύνθετη συνθήκη στην ενότητα Ελεγχος_2. Ο έλεγχος σωστά γίνεται με σύνθετη συνθήκη, παρά τα όσα αντίθετα έχουν γραφτεί. Εδώ γίνεται μια ακίνδυνη καταστρατήγηση του κανόνα για λόγους συντομίας. Πραγματικά στο συγκεκριμένο σημείο δεν πρόκειται να υπάρξει καμία αλλαγή, σε οποιαδήποτε επέκταση του προγράμματος.

Το παραπάνω πρόγραμμα χρησιμοποιείται απαραιτήτως ως υποπρόγραμμα από οποιοδήποτε πρόγραμμα επεξεργάζεται ημερομηνίες που εισάγονται από το πληκτρολόγιο. Ας σημειωθεί ότι για τον έλεγχο της αποδεκτής ημερομηνίας χρειάστηκαν αρκετές γραμμές. Αυτή θα είναι η επιπλέον επιβάρυνση κάθε τέτοιου προγράμματος.

14.1.4 Ταχύτητα

Ένα πρόγραμμα είναι ταχύτερο κάποιου άλλου, αν λύνει το ίδιο πρόβλημα σε λιγότερο χρόνο. Αυτό μπορεί να επιτυγχάνεται είτε:

- με αξιοποίηση πληροφοριών για την ποιότητα των δεδομένων. Για παράδειγμα ο αλγόριθμος ταξινόμησης, που είναι γενικά ταχύτερος (quick

sort), για ορισμένες ακραίες περιπτώσεις (περίπου ταξινομημένα στοιχεία) γίνεται βραδύτερος από άλλους,

- με διάφορες τεχνικές, όπως συμβαίνει με την περίπτωση των ψηφίων ελέγχου όπου δεν αλλάζει ο αλγόριθμος, αλλά ένα επιπλέον τμήμα που προστίθεται στην αρχή της λύσης, έχει ως αποτέλεσμα τη συνολική αύξηση της ταχύτητας του προγράμματος,
- με διαφορετικούς αλγόριθμους, όπως συμβαίνει με τους αλγόριθμους ταξινόμησης ή αναζήτησης, όπου για τα ίδια δεδομένα κάποιοι αλγόριθμοι αποδεικνύονται ταχύτεροι άλλων.
- με χρήση περισσότερης μνήμης

Το τελευταίο ξεφεύγει από τα όρια του βιβλίου, ενώ η τρίτη περίπτωση έχει αναπτυχθεί διεξοδικά στα κεφάλαια αλγοριθμικής και προγραμματισμού, επομένως θα δώσουμε παραδείγματα για τις δύο άλλες περιπτώσεις.

Παράδειγμα 5

Έστω ότι έχουμε να συγκρίνουμε μια σειρά αριθμών που εισάγονται από το πληκτρολόγιο και να απαντήσουμε πόσοι από αυτούς είναι αρνητικοί, θετικοί και μηδέν.

Προφανώς το πρόβλημα λύνεται με δύο εμφωλευμένα **ΑΝ** (τρεις έξοδοι).

Οι παρακάτω λύσεις είναι όλες αποδεκτές.

```
ΑΝ number>0 ΤΟΤΕ
    θ <- θ+1
ΑΛΛΙΩΣ_ΑΝ number<0 ΤΟΤΕ
    α <- α+1
ΑΛΛΙΩΣ
    μ <- μ+1
ΤΕΛΟΣ_ΑΝ

ΑΝ number<0 ΤΟΤΕ
    α <- α+1
ΑΛΛΙΩΣ_ΑΝ number>0 ΤΟΤΕ
    θ <- θ+1
ΑΛΛΙΩΣ
    μ <- μ+1
ΤΕΛΟΣ_ΑΝ
```

```
ΑΝ number=0 ΤΟΤΕ
    μ <- μ+1
ΑΛΛΙΩΣ_ΑΝ number<0 ΤΟΤΕ
    α <- α+1
ΑΛΛΙΩΣ
    θ <- θ+1
ΤΕΛΟΣ_ΑΝ
```

Ποια θα πρέπει να θεωρήσουμε ταχύτερη λύση;

Τέτοια ερώτηση δεν έχει απάντηση σε μία γενική θεώρηση, αν όμως μπορούμε να έχουμε την πληροφορία, του πως περίπου θα είναι τα δεδομένα μας, τότε πραγματικά μπορούμε να διαλέξουμε μια από τις τρεις που θα είναι η ταχύτερη για τα συγκεκριμένα δεδομένα. Για παράδειγμα αν γνωρίζουμε ότι στα δεδομένα μας οι περισσότεροι αριθμοί είναι μικρότεροι του μηδενός τότε η $2^η$ λύση θα δώσει ταχύτερα το τελικό αποτέλεσμα.

Παράδειγμα 6

Δίνεται πίνακας 1000 κωδικών αριθμών. Ζητείται να σχεδιαστεί πρόγραμμα που θα δέχεται έναν κωδικό από το πληκτρολόγιο, θα τον αναζητά στον πίνακα και θα δίνει τη θέση του πίνακα που βρέθηκε ο κωδικός.

Η αναζήτηση σε ένα πίνακα είναι θέμα το οποίο έχει ήδη αναπτυχθεί στο κεφάλαιο 3. Έτσι εδώ δεν θα λύσουμε το πρόβλημα, αλλά θα αναπτύξουμε μία ιδέα για την βελτίωση του χρόνου επεξεργασίας, που μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση. Είτε δηλαδή ο πίνακας είναι ταξινομημένος πίνακας και κάνουμε δυαδική αναζήτηση, είτε είναι μη ταξινομημένος και κάνουμε σειριακή αναζήτηση.

Η ιδέα έχει τη βάση της στις εξής παρατηρήσεις:

- Η αναζήτηση σε πίνακα είναι ούτως ή άλλως χρονοβόρα.
- Αν αναζητηθεί κωδικός που δεν υπάρχει στον πίνακα, το πρόγραμμα θα εκτελέσει όλα τα βήματα πριν το διαπιστώσει.
- Ο κωδικός που δεν υπάρχει, συνήθως είναι αποτέλεσμα λανθασμένης πληκτρολόγησης.

Επομένως αν αναπτυχθεί μία μέθοδος που θα ελέγχει τους κωδικούς που πληκτρολογούνται και θα επιτρέπει την αναζήτηση μόνο για τους κωδικούς που είναι σωστοί, αποδεκτοί, θα έχουμε μια σημαντική εξοικονόμηση του συνολικού χρόνου επεξεργασίας.

Οι κωδικοί με τους οποίους μπορούμε να πετύχουμε αυτόν τον έλεγχο περιέχουν ένα ακόμα ψηφίο το **ψηφίο ελέγχου** και παράγονται με ειδικό τρόπο. Πώς παράγεται ένα ψηφίο ελέγχου κωδικού;

Έστω ότι έχουμε πενταψήφιους κωδικούς και ένας από αυτούς είναι ο 12345. Ένας τρόπος για την παραγωγή του 6ου ψηφίου είναι ο εξής:

$$\Psi_6 = (\Psi_1*1+\Psi_2*3+\Psi_3*5+\Psi_4*7+\Psi_5*13) \bmod 11$$
$$=(1*1+2*3+3*5+4*7+5*13) \bmod 11 = (1+6+15+28+65) \bmod 11 = 5.$$

Άρα ο κωδικός που θα πληκτρολογείται θα είναι ο 123455. Με τον ίδιο κανόνα το πρόγραμμα θα ελέγχει, μετά την πληκτρολόγηση, αν ο εισαγόμενος 6ψήφιος, πλέον, κωδικός είναι αποδεκτός και μετά θα προχωρά στην αναζήτησή του στον πίνακα των 1000 θέσεων.

Ο κανόνας δημιουργίας του 6ου ψηφίου είναι ο εξής: Πολλαπλασιάζονται όλα τα ψηφία του κωδικού με έναν πρώτο αριθμό, και προστίθονται τα γινόμενα. Το άθροισμα διαιρείται με 11. Το υπόλοιπο της διαίρεσης είναι το ψηφίο ελέγχου. Το υπόλοιπο της διαίρεσης με το 11 είναι από 0 μέχρι 10. Αν βρεθεί υπόλοιπο 10, το 6ο ψηφίο είναι 0.

Υπάρχουν πολλοί τρόποι για τη δημιουργία του ψηφίου ελέγχου με πλεονεκτήματα και μειονεκτήματα.

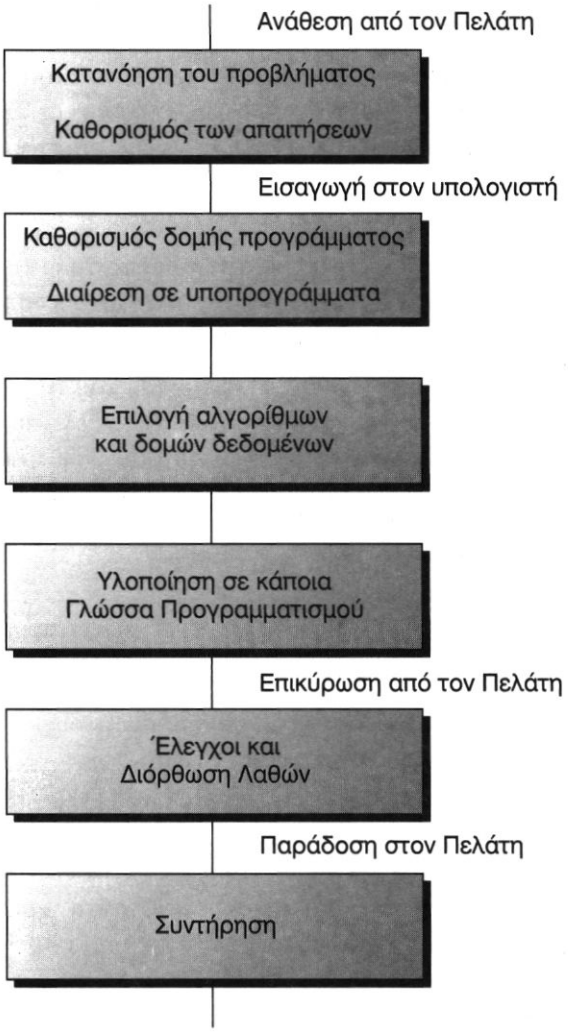
Το ψηφίο ελέγχου λοιπόν χρησιμοποιείται για λόγους ταχύτητας, όταν ο εισαγόμενος κωδικός πληκτρολογείται. Πράγματι ο χρόνος που χρειάζεται για να γίνουν οι απαραίτητες πράξεις για τον έλεγχο του κωδικού, ώστε αυτός να απορριφθεί, είναι κατά πολύ μικρότερος από το το χρόνο που χρειάζεται το πρόγραμμα για να κάνει 1000 συγκρίσεις, πριν καταλήξει στο συμπέρασμα ότι, ο κωδικός είναι λανθασμένος και φυσικά δεν υπάρχει.

Η μέθοδος εφαρμόζεται στους κωδικούς τραπεζικών λογαριασμών, στον κωδικό ΑΦΜ, στον κωδικό ISBN και γενικά όπου οι κωδικοί πληκτρολογούνται και η αναζήτηση τους πρόκειται να γίνει σε πολύ μεγάλους πίνακες ή αρχεία.

14.2 Τεκμηρίωση του Προγράμματος

Με τον όρο τεκμηρίωση εννοείται το σύνολο του γραπτού υλικού που περιγράφει τα συστατικά μέρη και τις λειτουργίες ενός νέου προγράμματος ή τις τροποποιήσεις που έγιναν σε ένα υπάρχον πρόγραμμα.

Στο σχήμα 14.1 φαίνεται η διαδοχή των φάσεων ανάπτυξης ενός προγράμματος.



Σχ. 14.1. *Φάσεις ανάπτυξης προγράμματος*

Η τεκμηρίωση δεν αποτελεί μια ξεχωριστή φάση στην ανάπτυξη ενός προγράμματος, αλλά μία παράλληλη διαδικασία που συμπληρώνει όλα τα στάδια ανάπτυξης ενός προγράμματος.

Οι παλιοί προγραμματιστές λένε: "Η τεκμηρίωση είναι χάσιμο χρόνου." Παραλείπουν όμως να πουν πόσο χρόνο αφιέρωσαν για να κάνουν μία αλλαγή στο παλιό τους πρόγραμμα που δεν είχε τεκμηρίωση.

Ας δούμε λοιπόν επιγραμματικά, ποιοι είναι οι λόγοι για τους οποίους χρειάζεται να γίνεται τεκμηρίωση, ποιες κατηγορίες τεκμηρίωσης υπάρχουν και τέλος ποιες ανάγκες εξυπηρετεί η ύπαρξη της τεκμηρίωσης.

14.2.1 Λόγοι τεκμηρίωσης

- **Παρόμοια εργασία.** Αν το πρόγραμμα που ζητείται, περιέχει παρόμοιες λειτουργίες με άλλα που έχουν ήδη δημιουργηθεί, τότε χρησιμοποιώντας την τεκμηρίωση, μπορούν να γίνουν οι απαραίτητες προσαρμογές σε αυτά.
- **Κάθε τροποποίηση γίνεται ευκολότερα.** Η ύπαρξη τεκμηρίωσης υποβοηθά στην επιτυχή και γρήγορη επέμβαση στο τμήμα του προγράμματος που πρέπει να μεταβληθεί ή να αντικατασταθεί ή τέλος να προστεθεί μια νέα λειτουργία στη σωστή θέση.
- **Δεν μπορεί να γίνει έλεγχος της λειτουργίας του αλγόριθμου.** Η απουσία τεκμηρίωσης ουσιαστικά απαγορεύει τον έλεγχο του αλγόριθμου. Αντίθετα η ύπαρξη του αρχείου δοκιμής (test file), μαζί με τα αποτελέσματα και τα σχόλια που το συνοδεύουν, θα περιορίσει έναν νέο έλεγχο μόνο στα σημεία που χρειάζεται.

14.2.2 Κατηγορίες τεκμηρίωσης

- **Τεκμηρίωση ανάπτυξης.** Αναφέρεται στις προδιαγραφές του προβλήματος, τι πρόκειται να κάνει το πρόγραμμα και τη σύνδεσή του με άλλα προγράμματα, αν υπάρχει. Περιέχει επίσης την περιγραφή των χρησιμοποιούμενων αλγορίθμων.
- **Τεκμηρίωση ελέγχου.** Αναφέρεται στα δεδομένα ελέγχου (test file) που χρησιμοποιήθηκαν προκειμένου να δοκιμαστεί το πρόγραμμα, αν έχουν διερευνηθεί οι ακραίες περιπτώσεις και ποιες είναι αυτές και τέλος αν έχουν καθιερωθεί κάποιες και ποιες συμβάσεις για τη λύση του προβλήματος. Κάθε πρόγραμμα έχει κάποια όρια, τα οποία πρέπει να αναφέρονται στην τεκμηρίωση.
- **Τεκμηρίωση χρήστη**. Περιέχει όλες τις οδηγίες που πρέπει να δοθούν στο χρήστη του προγράμματος προκειμένου να το χρησιμοποιήσει αποδοτικά. Οι οδηγίες αυτές συνήθως αναφέρονται ως εγχειρίδιο χρήστη (users manual).
- **Τεκμηρίωση προγράμματος.** Αναφέρεται στον τρόπο με τον οποίο έχουν λυθεί τα επιμέρους προβλήματα, τις υποθέσεις που έχουν γίνει και τους περιορισμούς που έχουν τεθεί. Ακόμα πληροφορίες για "ειδικές" τεχνικές που έχουν χρησιμοποιηθεί. Πρέπει σε κάθε βήμα να αναφερθεί με λεπτομέρεια, πολλές φορές κουραστική, αλλά δυστυχώς αναγκαία, η λύση που χρησιμοποιήθηκε και τα δεδομένα με τα οποία ελέγχθηκε. Ένα υπόμνημα των μεταβλητών που χρησιμοποιούνται είναι χρήσιμο για την τεκμηρίωση αλλά και για την αρχική ανάπτυξη του αλ-

γόριθμου. Η απλούστερη και πλέον στοιχειώδης τεκμηρίωση αυτής της μορφής γίνεται με την εισαγωγή γραμμών σχολίων μέσα στον κώδικα του προγράμματος.

Η τεκμηρίωση εξυπηρετεί λειτουργίες όπως:

- **Επικοινωνία μεταξύ χρήστη - αναλυτή - προγραμματιστή**. Στη φάση της κατανόησης του προβλήματος πολλές ερωτήσεις γίνονται, ενώ δίδονται οι αντίστοιχες απαντήσεις. Οι απαντήσεις αυτές αποτελούν τις οδηγίες βάση των οποίων θα γίνει το πρόγραμμα, επομένως είναι δεσμευτικές και για τα δύο μέρη, άρα είναι χρήσιμο να καταγράφονται. Η τεκμηρίωση πραγματοποιείται σε όλα τα στάδια της ανάπτυξης ενός προγράμματος. Κάθε διευκρίνιση που έχει δοθεί σε ερωτήματά μας, σε όλα τα στάδια ανάπτυξης πρέπει να βρίσκεται στο φάκελο τεκμηρίωσης.
- **Ιστορική αναφορά για τροποποιήσεις διορθώσεις**. Μια ιστορική αναφορά θα προσφέρει τη δυνατότητα συνολικής εκτίμησης του έργου που έχει γίνει.
- **Ποιοτικό έλεγχο.** Κάνει δυνατό έναν έλεγχο και δεν σπαταλάται χρόνος για τη γενική αναζήτηση του λάθους. Αν ακολουθήθηκε μία ευφυής μέθοδος, ποια είναι αυτή, καθώς και ποια είναι τα δεδομένα ελέγχου με τα οποία δοκιμάστηκε.
- **Αναφορά για διδακτικούς λόγους**. Έχοντας καταγράψει τη διαδρομή της σκέψης μας για το συγκεκριμένο αλγόριθμο, στη διάρκεια ζωής του προγράμματος μπορεί να επινοηθεί ένας άλλος καλύτερος τρόπος λύσης του ίδιου προβλήματος, λόγω της εμπειρίας που στο μεταξύ θα έχει αποκτηθεί. Να καταγράφεται η μέθοδος λύσης και ο λόγος που επιλέχθηκε. Να περιγράφεται το στιγμιότυπο στο οποίο βασίστηκε η λύση.

Παράδειγμα 7

Το μηχανογραφικό κέντρο του Υπουργείου Παιδείας πρόκειται να εκτυπώσει καταστάσεις με τις βαθμολογίες των πανελληνίων εξετάσεων που τελικά θα αποσταλούν στα σχολεία. Σε κάθε σελίδα αυτής της κατάστασης θα πρέπει να τυπώνεται η ονομασία του σχολείου. Όλες οι ονομασίες των σχολείων βρίσκονται σε ένα πίνακα από τον οποίο θα αναζητούνται. Σε κάθε γραμμή της σελίδας θα τυπώνεται το ονοματεπώνυμο, πατρώνυμο, και ο τελικός βαθμός του μαθητή.

Ένα τέτοιο πρόβλημα είναι φανερό ότι θα πρέπει να χωριστεί σε ενότητες προκειμένου να λυθεί. Για τη συντήρηση του προγράμματος είναι απα-

Το πρόβλημα του 2000 από τη σκοπιά της τεκμηρίωσης

Η ιδέα του να κρατά κανείς για το έτος δύο ψηφία, αντί τέσσερα που ήταν το ορθό, βρήκε πρόσφορο έδαφος και απλώθηκε παντού, σαν μία "έξυπνη" ιδέα που εξοικονομούσε χώρο μνήμης για το πρόγραμμα, αλλά και για την αποθήκευση των δεδομένων. 'Οπως φάνηκε όμως, δεν μελετήθηκαν όλες οι πιθανές επιπτώσεις, με αποτέλεσμα το κόστος της επαναφοράς των τεσσάρων ψηφίων για το έτος, να ανέλθει σε ιλιγγιώδη ποσά και χωρίς να είναι σίγουρο ότι βρέθηκαν όλες οι γραμμές κώδικα που έπρεπε να διορθωθούν. Αυτό το κόστος ήταν τόσο μεγάλο, γιατί είτε υπήρχαν παλιά προγράμματα που είχε χαθεί η τεκμηρίωση τους και άρα η επέμβαση στον κώδικα αφορούσε όλο το πρόγραμμα, είτε γιατί επειδή η "μέθοδος" ήταν πολύ διαδεδομένη, θεωρήθηκε ότι τέτοιες παρατηρήσεις δεν ήταν απαραίτητο να αναφέρονται στην τεκμηρίωση.

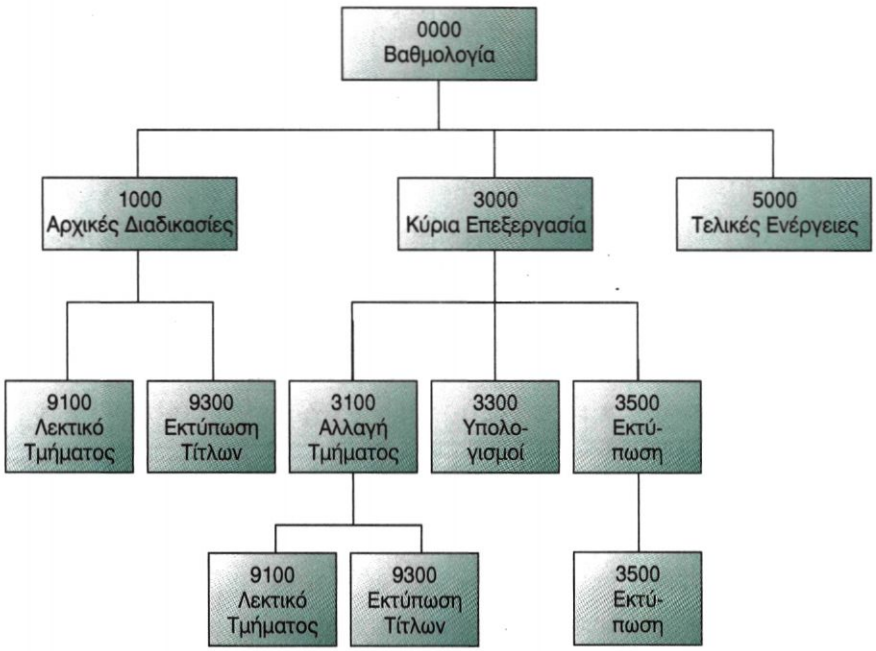
ραίτητο να έχει αποτυπωθεί η σύνδεση αυτών των ενοτήτων. Είναι προφανές ότι ο ψευδοκώδικάς του, θα εκτείνεται σε αρκετές σελίδες, όπου δύσκολα θα μπορεί κανείς να εντοπίσει τις διάφορες ενότητες, αλλά και τη σύνδεση που έχουν αυτές μεταξύ τους.

Για να καλυφθεί αυτή η ανάγκη είναι προτιμότερο να χρησιμοποιηθεί κάποια διαγραμματική τεχνική. Μία από τις πολλές που υπάρχουν είναι η HIPO (Hierarchy plus Input – Process –Output). Η τεχνική αυτή και χρησιμοποιεί διαγράμματα με τα οποία απεικονίζει την είσοδο, την έξοδο και τις λειτουργίες ενός προγράμματος. Απαρτίζεται από τρία είδη.

- Τον Οπτικό Πίνακα Περιεχομένων (Visual Table Of Contains, **VTOC**). Είναι ένα ιεραρχικό διάγραμμα αναπαράστασης των λειτουργιών του προγράμματος και της σχέσης μεταξύ τους.
- Το διάγραμμα Επισκόπησης Ιεραρχίας και Εισόδου – Επεξεργασίας - Εξόδου (Overview Hierarchy plus Input – Process – Output, **Overview HIPO**). Είναι διάγραμμα απεικόνισης εισόδου, διεργασιών και εξόδου, των λειτουργιών που εμφανίζονται στην κορυφή ενός οπτικού πίνακα περιεχομένων.
- Το Λεπτομερές Ιεραρχικό διάγραμμα Εισόδου, Επεξεργασίας, Εξόδου (**Detail HIPO**). Είναι διάγραμμα απεικόνισης εισόδου, λειτουργιών που εμφανίζονται στο χαμηλότερο τμήμα ενός οπτικού πίνακα περιεχομένων.

Από τα τρία διαγράμματα αυτό που είναι χρήσιμο για την τεκμηρίωση του προγράμματος, είναι το VTOC, τα δύο άλλα βρίσκουν εφαρμογή στο πεδίο της τεκμηρίωσης Πληροφοριακών Συστημάτων. 'Ενας οπτικός πίνακας περιεχομένων, είναι ένας μακροσκοπικός τρόπος αποτύπωσης των λειτουργιών του προγράμματος, που παρουσιάζει όχι τη λογική του, αλλά τα συστατικά του. Σ' αυτόν εμφανίζονται οι ενότητες από τις οποίες αποτελείται το πρόγραμμα, καθώς και ο ιεραρχικός τρόπος που σχετίζονται μεταξύ τους. 'Ετσι σε μία μόνο σελίδα αποτυπώνεται η εσωτερική διάρθρωση του προγράμματος, όση πολυπλοκότητα και αν παρουσιάζει.

Το VTOC είναι ένα σημαντικό εργαλείο για την καλή τεκμηρίωση του προγράμματός σας.



Σχ. 14.2. Διάγραμμα VTOC

Στο σχήμα 14.2 φαίνεται ότι το πρόγραμμα αποτελείται από οκτώ διαφορετικές ενότητες, που αναπτύσσονται σε τέσσερα επίπεδα. Η τοποθέτηση μιας ενότητας κάτω από μία άλλη σημαίνει ότι, η εκτέλεσή της εξαρτάται από αυτήν. Το βήμα που ακολουθεί αυτό το ιεραρχικό διάγραμμα, είναι η αναλυτική σχεδίαση κάθε ενότητας. Ας σημειωθεί ότι, ένας οπτικός πίνακας περιεχομένων διαβάζεται από πάνω προς τα κάτω και από αριστερά προς τα δεξιά.

Εκτός από το όνομα που δόθηκε σε κάθε ενότητα και που πρέπει να είναι ενδεικτικό της λειτουργίας της, δίνεται και ένας, συνήθως τετραψήφιος αριθμός. 'Ετσι στο μηδενικό επίπεδο που υπάρχει πάντοτε (ενότητα του προβλήματος), δίδεται ο αριθμός 0000. Στην πρώτη ενότητα του πρώτου επιπέδου δίδεται ο αριθμό 1000, στην επόμενη αυτής ο αριθμό 3000 κ.ο.κ.Ο αριθμός 9000 κρατείται για τις ενότητες που καλούνται από διάφορα επίπεδα και από διαφορετικούς κλάδους (κλάδος 1000, κλάδος 3000 κλπ.). Αυτές χαρακτηρίζονται ως ενότητες κοινής χρήσης, και είναι συνήθως η ενότητα της ανάγνωσης και των τίτλων.

Μετά την οριστικοποίηση των ενοτήτων πρώτου επιπέδου, κάθε ενότητα των άλλων επιπέδων θα συνοδεύεται από ένα αριθμό που θα είναι της ενότητας που την καλεί. 'Ετσι θα έχουμε τις ενότητες με αριθμούς 1100 και 1300 που θα καλούνται από την ενότητα 1000 κ.λπ.

Παρατηρήστε ότι οι αριθμοί που προηγούνται και συμπληρώνουν τα ονόματα των ενοτήτων έχουν ένα πολύ σημαντικό ρόλο. Δείχνουν μέσα στον κώδικα, αμέσως, την θέση της ενότητας στο πρόγραμμα, δηλαδή, από ποιες καλείται, ποιες καλεί και από ποιες είναι ανεξάρτητη. 'Ετσι σε μία δεδομένη επέμβαση γνωρίζετε που πρέπει να επέμβετε, αλλά και ποιες ενότητες πρέπει να ελέγξετε, μήπως επηρεάστηκαν από την επέμβασή σας.

Παρατηρείστε ακόμη ότι, σε κάθε κλάδο και σε κάθε επίπεδο η αρίθμηση δεν είναι συνεχόμενη. Αυτό σας δίνει την δυνατότητα σε μία μελλοντική συντήρηση, αν διαπιστώσετε ότι χρειάζεστε μια νέα ενότητα, να την τοποθετήσετε στη σωστή θέση με το σωστό αριθμό, χωρίς να χρειαστεί να επαναριθμήσετε όλες τις άλλες.

Κανόνας 3Ε

- *Είσοδος*
- *Επεξεργασία*
- *Έξοδος*

Σημειώνεται ότι

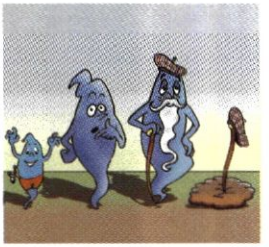
- Ο βαθμός ανεξαρτησίας καθορίζει το αν και πόσο οι αλλαγές σε μία ενότητα, επιβάλλουν αλλαγές σε άλλες ενότητες.
- Κάθε ενότητα έχει μία είσοδο, μία έξοδο και εκτελεί μία καθορισμένη επεξεργασία (κανόνας 3 Ε, **ε**ίσοδος, **ε**πεξεργασία, **έ**ξοδος).
- Κάθε ενότητα καλείται από μία άλλη ενότητα και επιστρέφει σ' αυτήν όταν τελειώσει.
- Κάθε ενότητα αποτελείται από τις βασικές δομές, ακολουθία, επιλογή και επανάληψη.
- Σε κάθε ενότητα δεν πρέπει να υπάρχει απότομη διακοπή, άπειρη επανάληψη.
- Πολλές ενότητες κάνουν το πρόγραμμα πολύπλοκο.

14.2.3 Φάκελλος Προγράμματος

Η ολοκλήρωση του προγράμματος σημαίνει την ώρα που σταματά η καθημερινή απασχόληση με αυτό. Η εργασία που έχει γίνει μέχρι τώρα, αλγόριθμοι, κωδικοποίηση, δεδομένα ελέγχου και αποτελέσματα, σχόλια από την πρώτη στιγμή που το πρόβλημα διατυπώθηκε μέχρι τώρα, διευκρινήσεις που ζητήθηκαν, απαντήσεις που δόθηκαν κάθε στιγμή της ανάπτυξης του προγράμματος, πρέπει να αρχειοθετηθούν. Συγκροτείται έτσι ο φάκελος τεκμηρίωσης, ο φάκελος του προγράμματος.

Ο φάκελος τεκμηρίωσης πρέπει να ολοκληρωθεί αμέσως μετά την ολοκλήρωση του προγράμματος. Η επόμενη φορά που θα γίνει προσπέλαση στο φάκελο αυτό, θα είναι όταν πρέπει να διαβαστεί για να αντληθούν οι απαραίτητες πληροφορίες για τη συντήρηση του προγράμματος.

14.3 Κύκλος Ζωής Λογισμικού



'Ενα πρόγραμμα αρχίζει την ζωή του από την στιγμή που θα καθοριστούν οι απαιτήσεις του, οι προδιαγραφές του και παύει να ζει όταν εξαντληθούν όλα τα περιθώρια συντήρησής του (προσθήκες, αλλαγές, βελτιώσεις).

Θα έχετε προσέξει ότι στα διάφορα πακέτα λογισμικού, είτε είναι γλώσσες προγραμματισμού, είτε πακέτα εφαρμογών, είτε λειτουργικά συστήματα, δίπλα στο εμπορικό όνομα του λογισμικού, υπάρχει ο αριθμός της έκδοσής του (version).

Ο αριθμός έκδοσης που συνοδεύει την ονομασία κάθε πακέτου λογισμικού, δείχνει ακριβώς τις αλλαγές που έχουν πραγματοποιηθεί από την αρχική του εμφάνιση. Ο κανόνας λέει ότι, όταν οι αλλαγές είναι σημαντικές, δηλαδή έχουν προστεθεί νέες λειτουργίες, εντολές, προγράμματα, ο αριθμός αυξάνει κατά ακέραιο αριθμό (DOS ver.5 DOS ver.6), όταν οι αλλαγές είναι μικρότερες, τότε αυξάνεται κατά δέκατα (Windows v.3.1, Windows v.3.11). Τα τελευταία χρόνια αυτή η εξέλιξη των δυνατοτήτων ενός λογισμικού έχει συνδεθεί με την πολιτική πωλήσεων των εταιρειών παραγωγής. 'Ετσι οι αριθμοί που ακολουθούν τις ονομασίες του λογισμικού, έχουν έντονη την χροιά του τμήματος πωλήσεων και όχι του τμήματος ανάπτυξης της εταιρείας (Windows 95, Windows98, Office 97 κ.λπ).

Θα έχετε παρατηρήσει ακόμα ή θα έχετε ακούσει, ότι το τάδε λογισμικό αντικαταστάθηκε από το δείνα, για παράδειγμα το MS-DOS αντικαταστάθηκε από τα Windows.

Οι αριθμοί έκδοσης και οι αντικαταστάσεις του λογισμικού δείχνουν με τον καλύτερο τρόπο ότι, κάθε λογισμικό έχει ένα κύκλο ζωής, όπου στον κύκλο αυτό γεννιέται και πεθαίνει όπως ένας ζωντανός οργανισμός.

Τι σημαίνει όμως "κύκλος ζωής"; Ποιες είναι οι ενδιάμεσες φάσεις του;

Μια σύγκριση θα μας δώσει καλύτερα το περιεχόμενο του όρου "Κύκλος Ζωής". Ας δούμε πρώτα τον "κύκλο ζωής" ενός αυτοκινήτου.

Η δημιουργία ενός αυτοκινήτου ξεκινά συνήθως από μία έρευνα που θα καταγράψει τις επιθυμίες του κοινού, αυτοκίνητο πόλης, εκτός δρόμου, φθηνό, γρήγορο και την τάση της αγοράς. Τα αποτελέσματα της έρευνας, σε συνδυασμό με άλλα δεδομένα, όπως άλλα μοντέλα του εργοστασίου, μοντέλα του ανταγωνισμού κ.λπ, θα καθορίσουν τις προδιαγραφές του νέου αυτοκινήτου, μικρό ή μεγάλο, γρήγορο, νεανικό, με πόσες πόρτες, με ποια πρόσθετα, σε ποια τιμή κ.λπ. (φάση ανάλυσης).

Οι μηχανικοί της εταιρείας έχοντας υπ' όψη τις προδιαγραφές θα σχεδιάσουν το νέο αυτοκίνητο επιλέγοντας τα κατάλληλα υλικά, δίνοντάς του την εξωτερική μορφή και τα άλλα χαρακτηριστικά που έχουν προδιαγραφεί, ιπποδύναμη, κατανάλωση, κ.λπ. (φάση σχεδιασμού).

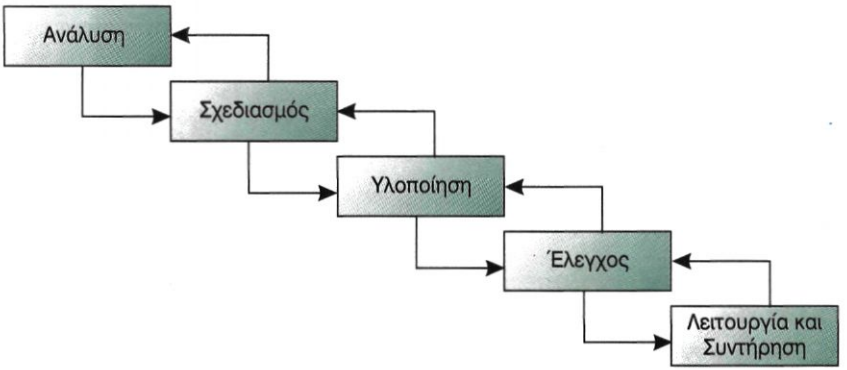
Το τμήμα παραγωγής του εργοστασίου θα πάρει τα σχέδια και θα προσαρμόσει τη γραμμή παραγωγής από πλευράς διαδικασιών και ελέγχων, ώστε να μπορεί να παράγεται το νέο αυτοκίνητο (φάση υλοποίησης).

Το έτοιμο αυτοκίνητο θα πρέπει να περάσει ένα τελικό έλεγχο λειτουργίας στο δρόμο και κάτω από ακραίες συνθήκες. Αυτός ο έλεγχος θα διενεργηθεί μέσα στις εγκαταστάσεις της εταιρείας, ώστε το νέο αυτοκίνητο να δοκιμαστεί σαν σύνολο (φάση ελέγχου).

Το δοκιμασμένο αυτοκίνητο είναι έτοιμο για πώληση. Πωλούμενο μπαίνει πλέον σε κανονική λειτουργία, η οποία εξασφαλίζεται με την κατάλληλη συντήρηση από τους εξουσιοδοτημένους μηχανικούς (φάση λειτουργίας και συντήρησης).

Αυτός λοιπόν είναι ο κύκλος ζωής ενός αυτοκινήτου, παρόμοιος σε αρκετά σημεία με τον κύκλο ζωής ενός προγράμματος, που παρουσιάζεται διαγραμματικά στο σχήμα 14.3.

Τα βέλη δείχνουν την αλληλεπίδραση των διαδοχικών φάσεων του κύκλου ζωής ενός προγράμματος. Δηλαδή ότι το τέλος μιας φάσης οδηγεί στην επόμενη, αλλά μπορεί να οδηγήσει και στην προηγούμενη. Όταν το τέλος μιας φάσης οδηγεί στην προηγούμενη, σημαίνει ότι ορισμένα στοιχεία, χρειάζεται να επανακαθοριστούν.



Σχ. 14.3. Κύκλος ζωής προγράμματος

Στη φάση **Ανάλυσης και Σχεδίασης** που ακολουθεί τον ορισμό του προβλήματος από τον πελάτη, μπορούμε να διακρίνουμε τις εξής ενέργειες:

- ✓ Καταγράφονται αναλυτικά τα δεδομένα και τα ζητούμενα του προβλήματος.
- ✓ Ζητούνται οι απαραίτητες διευκρινήσεις από τον πελάτη, σε όσα σημεία οι προδιαγραφές παρουσιάζουν ασάφεια.
- ✓ Καθορίζεται η δομή του προγράμματος.
- ✓ Καθορίζονται οι ενότητες (ρουτίνες, υποπρογράμματα) από τις οποίες θα αποτελείται το πρόγραμμα.
- ✓ Αναζητούνται έτοιμες ενότητες (modules) από παλιότερα προγράμματα που μπορούν να χρησιμοποιηθούν και σ' αυτό το πρόγραμμα.
- ✓ Επιλέγονται οι αλγόριθμοι και οι δομές δεδομένων που θα χρησιμοποιηθούν σε κάθε ενότητα.

Στην επόμενη φάση της **Υλοποίησης** του προγράμματος σε κάποια γλώσσα προγραμματισμού, ακολουθούμε τα εξής βήματα με τη σειρά:

- ✓ Επιλέγεται η γλώσσα προγραμματισμού για το συγκεκριμένο πρόγραμμα. Σημειώστε ότι όλες γλώσσες δεν είναι κατάλληλες για όλα τα προβλήματα.
- ✓ Εισάγεται το κωδικοποιημένο πρόγραμμα στον υπολογιστή. Το πρόγραμμα αυτό είναι το αρχικό πρόγραμμα (source program).
- ✓ Ζητείται η μετάφραση του προγράμματος από ένα μεταγλωττιστή, ώστε αυτό να γίνει κατανοητό από τον υπολογιστή. Το πρόγραμμα αυτό είναι το τελικό πρόγραμμα (object program).

Προσοχή: Ο έλεγχος ενός προγράμματος, διαπιστώνει την ύπαρξη ενός λάθους, ποτέ όμως δεν πιστοποιεί την ανυπαρξία λάθους.

- ✓ Η μετάφραση θα αποκαλύψει λάθη "ορθογραφίας" και συντακτικού" της γλώσσας προγραμματισμού.
- ✓ Διορθώνονται τα λάθη και ακολουθεί ξανά μετάφραση του προγράμματος, έως την οριστική εξάλειψή τους.

Η φάση των **Ελέγχων** αναφέρεται στον έλεγχο των λογικών λαθών που πιθανόν να υπάρχουν σε ένα πρόγραμμα. Η σειρά των ελέγχων είναι η ακόλουθη.

- ✓ Το πρόγραμμα ελέγχεται, με τα δεδομένα ελέγχου που είχε ελεγχθεί και ο αλγόριθμος, για να διαπιστωθεί αν παράγει τα επιθυμητά αποτελέσματα.
- ✓ Διαπιστώνονται λάθη που οφείλονται είτε σε λάθη κατά την κωδικοποίηση του αλγόριθμου είτε από λανθασμένη επικοινωνία ενοτήτων.
- ✓ Τα τυχόν λάθη διορθώνονται και οι έλεγχοι συνεχίζονται μέχρι το πρόγραμμα να απαλλαγεί από αυτά.
- ✓ Παρουσιάζονται τα αποτελέσματα στον πελάτη προκειμένου να έχουμε την οριστική επικύρωση εκ μέρους του.
- ✓ 'Αν παρουσιάζεται απόκλιση από τις προδιαγραφές, θα πρέπει να επιστρέψουμε στην φάση της ανάλυσης και σχεδίασης προκειμένου να γίνουν οι κατάλληλες διορθώσεις.

Στο τέλος της φάσης των ελέγχων έχουμε ένα έτοιμο πρόγραμμα που μπορούμε να παραδώσουμε στον πελάτη για χρήση. Δεν πρέπει να ξεχνάτε, ότι ο έλεγχος του προγράμματος διαπιστώνει την ύπαρξη ενός λάθους που πρέπει να διορθωθεί, δυστυχώς όμως δεν πιστοποιεί την απουσία λάθους. 'Ετσι ένα πρόγραμμα κατ' ουσία βρίσκεται κάτω από ένα διαρκή έλεγχο της ορθής λειτουργίας του.

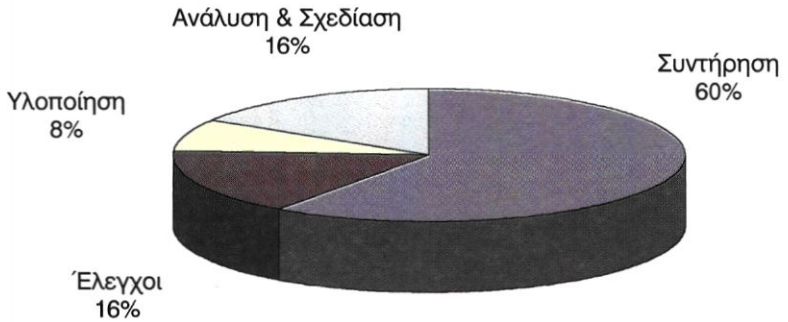
Στην επόμενη φάση της **Συντήρησης** θα γίνουν όλες οι προσαρμογές και βελτιώσεις που χρειάζονται προκειμένου το πρόγραμμα σας να συνεχίσει να χρησιμοποιείται. Η φάση αυτή διαρκεί όσο θα χρησιμοποιείται το πρόγραμμα σας. Εδώ θα παρατηρήσουμε ότι:

- ✓ Οι προσαρμογές είναι αναπόφευκτες όταν διαφοροποιούνται τα δεδομένα του προβλήματος ή όταν ο χρήστης ζητήσει νέες λειτουργίες.
- ✓ Κάποιες προσαρμογές μπορεί να απαιτήσουν την εκτέλεση της φάσης της ανάλυσης και σχεδίασης και άρα όλων των υπολοίπων φάσεων.
- ✓ Οι βελτιώσεις προκύπτουν από την εμπειρία που αποκτάται με τον καιρό και μας κάνει να "βλέπουμε" τα ίδια πράγματα με "άλλο μάτι".

- ✓ Κάθε προσαρμογή ή βελτίωση θα πρέπει να καταλήγει σε συνολικό έλεγχο του προγράμματος και φυσικά στην καταγραφή των σχετικών σχολίων για την τεκμηρίωση.
- ✓ Το τελευταίο στάδιο αυτής της φάσης έρχεται με την τελειοποίηση του προγράμματος. Τώρα το πρόγραμμά σας δουλεύει υποδειγματικά ... μέχρι την επόμενη αλλαγή.

Στο σχήμα 14.4 παρουσιάζεται η προσπάθεια που απαιτείται κατά τις διάφορες φάσεις του κύκλου ζωής ως ποσοστό του συνολικού έργου.

Κύκλος Ζωής Προγράμματος



Σχ. 14.4. *Κατανομή της προσπάθειας που απαιτείται στις διάφορες φάσεις του κύκλου ζωής προγράμματος*

Ανακεφαλαίωση

- Μάθαμε να απορρίπτουμε λύσεις που δεν πληρούν τα βασικά χαρακτηριστικά αξιολόγησης προγράμματος.
- Είδαμε ότι το μεγαλύτερο μέρος της ζωής ενός προγράμματος διανύεται στην φάση συντήρησης του.
- Μάθαμε να εκτιμούμε την φιλικότητα ενός προγράμματος σαν χρήστες, αλλά και να διαπιστώνουμε τις δυσκολίες που έχουν τα φιλικά προγράμματα από πλευράς προγραμματισμού.
- Μάθαμε ότι πριν προχωρήσουμε στη λύση ενός προβλήματος πρέπει να γνωρίσουμε τον χώρο που ανήκει το πρόβλημα.
- Μάθαμε ποια είναι τα βήματα ανάπτυξης λογισμικού και σε ποια θέματα κατανέμεται ο χρόνος στον κύκλο ζωής ενός προγράμματος.

- Είδαμε πως μπορούμε να αυξήσουμε την ταχύτητα ενός αλγόριθμου, αξιοποιώντας πληροφορίες και τεχνικές. Για παράδειγμα με τη χρήση ψηφίων ελέγχου κωδικού δημιουργούμε ταχύτερα προγράμματα.
- Μάθαμε τι είναι το διάγραμμα VTOC και πως αυτό συμπληρώνει την τεκμηρίωση της λύσης ενός προβλήματος.

Λέξεις Κλειδιά

Κύκλος ζωής προγράμματος, Συντήρηση προγράμματος, HIPO, VTOC, Οπτικός Πίνακας Περιεχομένων, Δεδομένα ελέγχου.

Ερωτήσεις

1. Ποιες είναι οι φάσεις του κύκλου ζωής προγράμματος;
2. Σε ποια φάση της ανάπτυξης ενός προγράμματος αρχίζει η τεκμηρίωση και πότε τελειώνει;
3. Ποιος είναι ο κανόνας 3 Ε;
4. Ποια η σημασία του αριθμού που συνήθως ακολουθεί το όνομα ενός πακέτου λογισμικού;
5. Ποια η σημασία των αριθμών που συνοδεύουν το όνομα μιας ενότητας σε έναν οπτικό πίνακα περιεχομένων (VTOC).
6. Ποιο κριτήριο αξιολόγησης προγράμματος θεωρείτε ότι είναι σπουδαιότερο;
7. Ποιοι λόγοι επιβάλλουν την τεκμηρίωση του προγράμματος.
8. Τι εξυπηρετεί η τεκμηρίωση ενός προγράμματος;
9. Ποια στοιχεία περιέχει ο φάκελος προγράμματος;
10. Πιστεύετε ότι η συντήρηση του προγράμματος είναι σημαντική εργασία;
11. Είναι δυνατόν ένα πρόγραμμα να περιέχει λογικό λάθος και να λειτουργεί;

Βιβλιογραφία

1. *Εγκυκλοπαίδεια Πληροφορικής και Τεχνολογίας Υπολογιστών*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1986.
2. Β. Λαοπόδης, *Ανάλυση και Σχεδίαση Συστημάτων*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1992.

Διευθύνσεις Διαδικτύου

- http://www.acm.org/

Τα κριτήρια αξιολόγησης μέσα από την ιστοσελίδα του παλιότερου και μεγαλύτερου, σε παγκόσμιο επίπεδο, επιστημονικού και εκπαιδευτικού οργανισμού υπολογιστών, του ACM (Association of Computing Machinery).

ΚΕΦΑΛΑΙΟ 4ο

ΤΕΧΝΙΚΕΣ ΣΧΕΔΙΑΣΗΣ ΑΛΓΟΡΙΘΜΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ

- Ανάλυση προβλημάτων
- Μέθοδοι σχεδίασης αλγορίθμων
- Μέθοδος διαίρει και βασίλευε
- Δυναμικός προγραμματισμός
- Άπληστη μέθοδος

Εισαγωγή

Όταν μας δίνεται ένα πρόβλημα, και πριν ασχοληθούμε με την κωδικοποίηση και τον προγραμματισμό, πρέπει προκαταρκτικά να κάνουμε μία θεωρητική μελέτη του προβλήματος. Πιο συγκεκριμένα, μία καλή ιδέα είναι να προσπαθήσουμε να διαπιστώσουμε, αν το δεδομένο πρόβλημα μπορεί να αντιμετωπισθεί με βάση κάποιες γενικές μεθοδολογίες ανάπτυξης αλγορίθμων. Σκοπός του κεφαλαίου είναι η εισαγωγή στις μεθολογίες αυτές, που αποτελούν ένα σύνολο τεχνικών και βημάτων που πρέπει να ακολουθηθούν. Για κάθε μεθοδολογία θα δούμε παραδείγματα επίλυσης γνωστών και καθημερινών προβλημάτων.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές:

- να τεκμηριώνουν την αναγκαιότητα ανάλυσης των προβλημάτων και σχεδίασης των κατάλληλων αλγορίθμων,
- να μπορούν να καταγράφουν την ακολουθία βημάτων για την ανάλυση των αλγορίθμων,
- να διατυπώνουν σύγχρονες τεχνικές σχεδίασης αλγορίθμων,
- να περιγράφουν τις κυριότερες προσεγγίσεις επίλυσης και ανάλυσης προβλημάτων,
- να επιλύουν προβλήματα με χρήση των κυριοτέρων προσεγγίσεων.

Προερωτήσεις

- ✓ Έχεις ακούσει για το πρόβλημα του περιοδεύοντα πωλητή;
- ✓ Είχες ποτέ φανταστεί ότι το «διαίρει και βασίλευε» μπορεί να έχει εφαρμογή στην Πληροφορική;
- ✓ Μπορεί να βρεθεί η ρίζα μιας μη πολυωνυμικής εξίσωσης;

4.1 Ανάλυση προβλημάτων

Σύμφωνα με όσα αναφέρθηκαν στα προηγούμενα κεφάλαια, ο αλγόριθμος αποσκοπεί στην επίλυση ενός προβλήματος. Είναι πιθανόν ένα πρόβλημα να μην επιλύεται με μία μόνο λύση αλλά με περισσότερες. Γενικά, η λύση σε ένα πρόβλημα μπορεί να προέλθει από ποικίλες και διαφορετικές προσεγγίσεις, τεχνικές και μεθόδους. Έτσι, είναι απαραίτητο να γίνεται μία καλή ανάλυση του κάθε προβλήματος και να προτείνεται συγκεκριμένη μεθοδολογία και ακολουθία βημάτων. Βασικός στόχος μας είναι η πρόταση έξυπνων και αποδοτικών λύσεων.

Η ανάλυση ενός προβλήματος σε ένα σύγχρονο υπολογιστικό περιβάλλον περιλαμβάνει:

- ✓ την καταγραφή της υπάρχουσας πληροφορίας για το πρόβλημα,
- ✓ την αναγνώριση των ιδιαιτεροτήτων του προβλήματος,
- ✓ την αποτύπωση των συνθηκών και προϋποθέσεων υλοποίησής του

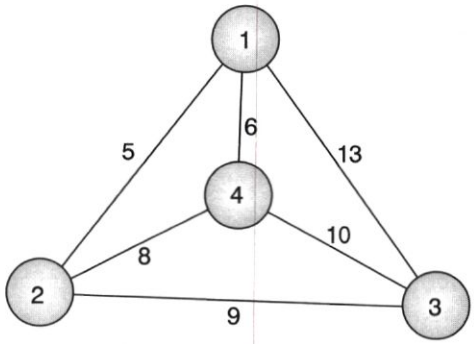
και στη συνέχεια:

- ✓ την πρόταση επίλυσης με χρήση κάποιας μεθόδου, και
- ✓ την τελική επίλυση με χρήση υπολογιστικών συστημάτων.

Έτσι, κατά την ανάλυση ενός προβλήματος θα πρέπει να δοθεί απάντηση σε κάθε μία από τις επόμενες ερωτήσεις:

1. Ποια είναι τα δεδομένα και το μέγεθος του προβλήματος,
2. Ποιες είναι οι συνθήκες που πρέπει να πληρούνται για την επίλυση του προβλήματος,
3. Ποια είναι η πλέον αποδοτική μέθοδος επίλυσής τους (σχεδίαση αλγορίθμου),
4. Πώς θα καταγραφεί η λύση σε ένα πρόβλημα (π.χ. σε ψευδογλώσσα), και
5. Ποιος είναι ο τρόπος υλοποίησης στο συγκεκριμένο υπολογιστικό σύστημα (π.χ. επιλογή γλώσσας προγραμματισμού)

Παράδειγμα. Έστω ότι αντιμετωπίζουμε το πρόβλημα ενός ταχυδρομικού διανομέα, που πρέπει να ξεκινήσει από ένα χωριό, να επισκεφθεί έναν αριθμό από γειτονικά χωριά, για να μοιράσει ένα σύνολο επιστολών και να επιστρέψει στο χωριό, από όπου ξεκίνησε περνώντας μόνο μία φορά από κάθε χωριό . Το πρόβλημα έγκειται στην εύρεση της καλύτερης διαδρομής, έτσι ώστε ο διανομέας να διανύσει το μικρότερο δυνατό αριθμό χιλιομέτρων.



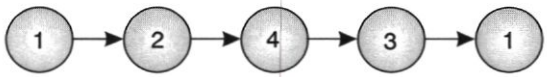
Σχ. 4.1. *Χωριά και αποστάσεις μεταξύ τους.*

Στο προηγούμενο σχήμα οι 4 αριθμημένοι κόμβοι αντιστοιχούν στα 4 χωριά, ενώ οι 6 συνδέσεις αντιστοιχούν στις οδικές αρτηρίες που ενώνουν τα χωριά αυτά. Τέλος, οι ακέραιοι που χαρακτηρίζουν τις συνδέσεις μεταξύ των κύκλων παρουσιάζουν τις αντίστοιχες χιλιομετρικές αποστάσεις μεταξύ των χωριών. Επιπλέον, ας υποθέσουμε ότι ο διανομέας ξεκινά από το χωριό 1, και βέβαια σε αυτό πρέπει να καταλήξει, αφού επισκεφθεί τα χωριά 2, 3 και 4. Στο πρόβλημα αυτό μπορούν να υπάρξουν διάφορες προσεγγίσεις για την ανάλυση και την επίλυσή του.

Α) Μία πρώτη ανάλυση του προβλήματος είναι:

1. να γίνει καταγραφή όλων των αποστάσεων μεταξύ των χωριών,
2. να ταξινομηθούν οι συνδέσεις των χωριών κατά αύξουσα χιλιομετρική απόσταση,
3. να επιλέγεται κάθε φορά η μετάβαση από το χωριό όπου βρίσκεται ο διανομέας προς το πλησιέστερο χωριό.

Με βάση τα παραπάνω βήματα ο διανομέας θα επέλεγε την εξής σειρά επίσκεψης των χωριών:



διανύοντας συνολικά 36 χιλιόμετρα.

Β) Μία διαφορετική ανάλυση του προβλήματος είναι:

1. να γίνει καταγραφή όλων των αποστάσεων μεταξύ των χωριών,
2. να βρεθεί μία σειρά επίσκεψης των χωριών με στόχο την ελαχιστοποίηση της συνολικής απόστασης και όχι την ελαχιστοποίηση της κάθε φορά απόστασης.

Με βάση τα παραπάνω βήματα ο διανομέας θα επέλεγε την εξής σειρά επίσκεψης των χωριών:

οπότε η συνολική διανυόμενη απόσταση είναι 30 χιλιόμετρα.

Επομένως είναι φανερό, ότι η ανάλυση κάθε προβλήματος είναι απαραίτητη, έτσι ώστε να αναζητηθεί η πλέον κατάλληλη μέθοδος που να παρέχει τη ζητούμενη λύση, όσο γίνεται ταχύτερα και με το λιγότερο δυνατό κόστος σε υπολογιστικούς πόρους. Δεν υπάρχει ένας ενιαίος κανόνας, μία γενική φόρμουλα που να αναφέρεται στην επίλυση του συνόλου των προβλημάτων. Υπάρχουν όμως "συγγενή" προβλήματα, δηλαδή προβλήματα που μπορούν να αναλυθούν με παρόμοιο τρόπο και να αντιμετωπισθούν με αντίστοιχες μεθόδους και τεχνικές. Γενικότερα, οι μέθοδοι ανάλυσης και επίλυσης των προβλημάτων παρουσιάζουν ιδιαίτερο ενδιαφέρον για τους εξής λόγους:

- παρέχουν ένα γενικό πρότυπο κατάλληλο για την επίλυση προβλημάτων ευρείας κλίμακας,
- μπορούν να αναπαρασταθούν με κοινές δομές δεδομένων και ελέγχου (που υποστηρίζονται από τις περισσότερες σύγχρονες γλώσσες προγραμματισμού),
- παρέχουν τη δυνατότητα καταγραφής των χρονικών και "χωρικών" απαιτήσεων της μεθόδου επίλυσης, έτσι ώστε να μπορεί να γίνει επακριβής εκτίμηση των αποτελεσμάτων.

4.2 Μέθοδοι σχεδίασης αλγορίθμων

Οι μέθοδοι λύσης ενός προβλήματος που προκύπτουν από την ανάλυσή του, οδηγούν στη σχεδίαση ενός αλγορίθμου που συνιστά την ακολουθία βημάτων, που πρέπει να ακολουθηθούν για να επιλυθεί το πρόβλημα. Όπως αναφέρθηκε, υπάρχει περίπτωση να παρουσιασθούν περισσότε-

ρες από μία τεχνικές για τη λύση ενός προβλήματος. Επομένως, για να προταθεί η καλύτερη λύση χρειάζεται να γίνουν κάποιες επιλογές και παραδοχές κατά τη διαδικασία σύνθεσης και σχεδίασης του αλγορίθμου.

Υπάρχουν μερικές τεχνικές που συχνά χρησιμοποιούνται σε πληθώρα προβλημάτων. Έτσι, οι τεχνικές αυτές έχουν τυποποιηθεί λόγω των κοινών χαρακτηριστικών τους κατά την επίλυση ενός προβλήματος. Η τυποποίηση αυτή σε κάποιο βαθμό διευκολύνει στην ένταξη κάποιου προβλήματος στην αντίστοιχη κατηγορία επίλυσής του (όπου αυτό είναι δυνατό). Γενικότερα, κάθε τεχνική χρειάζεται να υποστηρίζει τα εξής:

- ✓ να αντιμετωπίζει με το δικό της τρόπο τα δεδομένα,
- ✓ να έχει τη δική της ακολουθία εντολών και
- ✓ να διαθέτει τη δική της αποδοτικότητα.

Επομένως κάθε τεχνική έχει τα δικά της χαρακτηριστικά και τις δικές της ιδιαιτερότητες.

Κατά την επίλυση ενός προβλήματος, επιχειρείται σύγκριση των χαρακτηριστικών και των ιδιοτήτων των τεχνικών που μπορούν να αποτελέσουν πρόταση λύσης του προβλήματος. Το αποτέλεσμα της σύγκρισης των διαφορετικών τεχνικών είναι η επιλογή της καταλληλότερης τεχνικής για την επίλυση του συγκεκριμένου προβλήματος. Στη βιβλιογραφία αναφέρονται αρκετές τυποποιημένες κατηγορίες τεχνικών, όμως στα πλαίσια του βιβλίου αυτού θα εξετάσουμε μερικές μόνο από αυτές:

Δεν υπάρχει αλγόριθμος για τη σχεδίαση αλγορίθμων.

- ✓ Μέθοδος διαίρει και βασίλευε
- ✓ Μέθοδος δυναμικού προγραμματισμού
- ✓ Άπληστη μέθοδος

Κάθε ένα από αυτά τα είδη προσεγγίσεων θα εξετασθεί ιδιαιτέρως στη συνέχεια.

Παρ' ότι στη βιβλιογραφία αναφέρονται αρκετές τεχνικές σχεδίασης αλγορίθμων, αυτό δεν σημαίνει ότι κάθε πρόβλημα μπορεί να επιλυθεί εφαρμόζοντας μία από αυτές τις γνωστές τεχνικές. Υπάρχουν πολλά προβλήματα που για την επίλυσή τους απαιτούν την εφαρμογή μίας νέας αντίληψης. Συχνά, οι μέθοδοι που εφαρμόζονται στα προβλήματα αυτά ονομάζονται *ευριστικές τεχνικές*.

4.3 Μέθοδος διαίρει και βασίλευε

Στην κατηγορία *"Διαίρει και Βασίλευε"* (divide and conquer) εντάσσονται οι τεχνικές που υποδιαιρούν ένα πρόβλημα σε μικρότερα υποπροβλήματα, που έχουν την ίδια τυποποίηση με το αρχικό πρόβλημα αλλά είναι μικρότερα σε μέγεθος. Με όμοιο τρόπο, τα υπο-προβλήματα αυτά μπορούν να διαιρεθούν σε ακόμη μικρότερα υποπροβλήματα κοκ. Έτσι η επίλυση ενός προβλήματος έγκειται στη σταδιακή επίλυση των όσο το δυνατόν μικρότερων υποπροβλημάτων, ώστε τελικά να καταλήξουμε στη συνολική λύση του αρχικού ευρύτερου προβλήματος. Αυτή η προσέγγιση ονομάζεται από *επάνω προς τα κάτω* (top-down).

Πιο τυπικά, η περιγραφή αυτής της μεθόδου σχεδίασης αλγορίθμων μπορεί να αποδοθεί με τα επόμενα βήματα:

1. Δίνεται για επίλυση ένα στιγμιότυπο ενός προβλήματος.
2. Υποδιαιρείται το στιγμιότυπο του προβλήματος σε υπο-στιγμιότυπα του ίδιου προβλήματος.
3. Δίνεται ανεξάρτητη λύση σε κάθε ένα υπο-στιγμιότυπο.
4. Συνδυάζονται όλες οι μερικές λύσεις που βρέθηκαν για τα υπο-στιγμιότυπα, έτσι ώστε να δοθεί η συνολική λύση του προβλήματος.

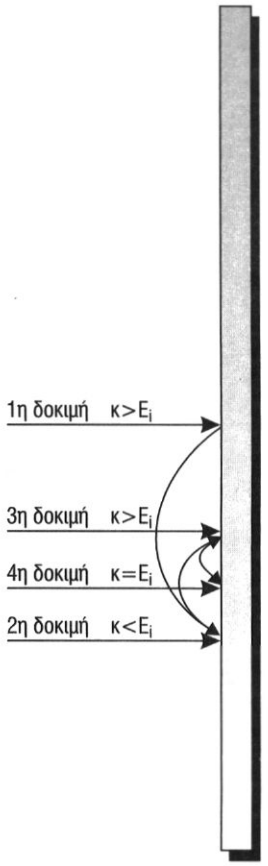
Στη συνέχεια παρουσιάζεται ένας κλασικός αλγόριθμος που ακολουθεί τη φιλοσοφία της μεθόδου Διαίρει και Βασίλευε: η Δυαδική αναζήτηση.

Δυαδική αναζήτηση

Έστω ότι δίνεται ένας ηλεκτρονικός τηλεφωνικός κατάλογος, δηλαδή ένας πίνακας με n ονόματα και έναν αριθμό τηλεφώνου για κάθε όνομα. Ο κατάλογος είναι ταξινομημένος κατά αύξουσα αλφαβητική τάξη ως προς τα ονόματα. Έστω ότι τα n ονόματα με τα αντίστοιχα τηλέφωνα είναι αποθηκευμένα σε δύο πίνακες: names[1..n] και phones[1..n] αντίστοιχα. Σε μία τέτοια περίπτωση, λοιπόν, το ζητούμενο είναι να βρεθεί το τηλέφωνο που αντιστοιχεί σε δεδομένο όνομα συνδρομητή (υποθέτοντας ότι το δεδομένο αναζητούμενο όνομα πράγματι υπάρχει στον κατάλογο).

Το πρόβλημα επιλύεται με τη μέθοδο Διαίρει και Βασίλευε με βάση την παρατήρηση ότι όταν δοθεί ένα όνομα, τότε υπάρχουν οι εξής 3 περιπτώσεις:

1. το όνομα βρίσκεται στη μεσαία θέση του πίνακα names,



Σχ. 4.2. Δυαδική αναζήτηση

2. το όνομα βρίσκεται σε μία θέση στο πρώτο μισό κομμάτι του πίνακα names, ή

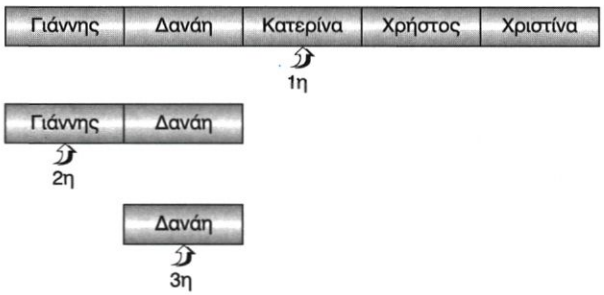
3. το όνομα βρίσκεται σε μία θέση στο δεύτερο μισό κομμάτι του πίνακα names.

Η δυαδική αναζήτηση χρησιμοποιείται μόνο στην περίπτωση ταξινομημένου πίνακα.

Η παρατήρηση αυτή οδηγεί στη σύνταξη του επόμενου αλγορίθμου. Σημειώνεται ότι ο αλγόριθμος αυτός ισχύει μόνο για την περίπτωση που το αναζητούμενο όνομα υπάρχει στον κατάλογο (δηλαδή, στον πίνακα names). Ο αλγόριθμος αυτός μπορεί εύκολα να τροποποιηθεί ώστε να αντιμετωπίζει και την περίπτωση της "ανεπιτυχούς αναζήτησης", δηλαδής της περίπτωσης όπου το αναζητούμενο όνομα δεν υπάρχει στον πίνακα.

```
Αλγόριθμος Δυαδική_αναζήτηση
Δεδομένα // names, phones, onoma, arxi, telos //
meso ← [arhi + telos]/2
Αν onoma = names[meso] τότε
    Tel ← phones[meso]
αλλιώς
    Αν onoma < names[meso] τότε
    Δυαδική_αναζήτηση(names, phones, onoma, arhi, meso-1)
    αλλιώς
    Δυαδική_αναζήτηση(names, phones, onoma, meso+1,telos)
    Τέλος_αν
Τέλος_αν
Αποτελέσματα  // Tel //
Τέλος Δυαδική_αναζήτηση
```

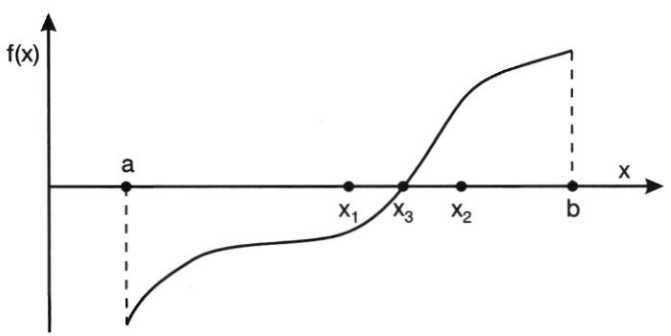
Παράδειγμα. Το παράδειγμα που ακολουθεί παρακολουθεί την αλληλουχία των βημάτων του αλγορίθμου της δυαδικής αναζήτησης ενός ονόματος σε έναν πίνακα πέντε θέσεων. Έστω ότι στον τηλεφωνικό κατάλογο αναζητείται το όνομα *Δανάη*. Κατά την πρώτη προσπάθεια από τις πέντε



***Σχ. 4.3** Διαδοχικές συγκρίσεις σε αρχικό πίνακα και υπο-πίνακες για την αναζήτηση του ονόματος Δανάη.*

θέσεις ελέγχεται η μεσαία, δηλαδή η τρίτη, όπου βρίσκεται το όνομα *Κατερίνα*. Το όνομα *Δανάη* είναι λεξικογραφικά μικρότερο από το όνομα *Κατερίνα* και επομένως η αναζήτηση περιορίζεται στο πρώτο μισό του πίνακα που αποτελείται από δύο θέσεις. Από τις δύο αυτές θέσεις, σε μία δεύτερη προσπάθεια ελέγχεται η πρώτη θέση, όπου βρίσκεται το τηλέφωνο του *Γιάννη*. Επειδή, το όνομα Δανάη είναι λεξικογραφικά μεγαλύτερο από το όνομα *Γιάννης*, η αναζήτηση περιορίζεται στον υπο-πίνακα που αποτελείται από τη δεύτερη θέση μόνο. Έτσι, στη θέση αυτή με μία τρίτη προσπάθεια βρίσκουμε το τηλέφωνο της *Δανάης*.

Ο τρόπος που λειτουργεί η δυαδική αναζήτηση είναι ανάλογος με αυτόν της μεθόδου της διχοτόμησης. Η *μέθοδος της διχοτόμησης* (ή του Bolzano) χρησιμοποιείται για την εύρεση μιας ρίζας της εξίσωσης $f(x)=0$ στο διάστημα $[a, b]$. Ως γνωστό, στο διάστημα αυτό υπάρχει μία τουλάχιστον ρίζα, αν ισχύει $f(a).f(b)<0$ (για την ακρίβεια μπορεί να υπάρχει περιττός αριθμός ριζών). Με τη μέθοδο της διχοτόμησης βρίσκεται ένα σημείο x_1 $(=(a+b)/2)$ στο μέσο του διαστήματος $[a, b]$ και εξετάζεται, αν $f(a).f(x_1)<0$. Αν ναι, τότε η ρίζα υπάρχει στο διάστημα $[a, x_1]$, αλλιώς στο $[x_1, b]$. Στη συνέχεια βρίσκεται το μέσο x_2 του υποδιαστήματος που υπάρχει η ρίζα και επαναλαμβάνονται τα ίδια. Ετσι σε κάθε επανάληψη εξαιρείται από την έρευνα το μισό διάστημα και προφανώς η ακολουθία τιμών x_1, x_2, ... συγκλίνει προς τη ρίζα της εξίσωσης. Η διαδικασία τερματίζεται, όταν η προσέγγιση της ρίζας θεωρείται ικανοποιητική.



***Σχ. 4.4** Σχηματική παράσταση της μεθόδου της διχοτόμησης*

4.4 Δυναμικός προγραμματισμός

Οι αλγόριθμοι της προηγούμενης παραγράφου στηρίζονται στη φιλοσοφία που συνιστά να σπάζουμε ένα πρόβλημα σε μικρότερα, ώστε να επιλύονται ευκολότερα. Έτσι η επίλυση του αρχικού προβλήματος έρχεται στη συνέχεια ως σύνθεση της επίλυσης των μικρότερων υπο - προβλημά-

των. Στην παράγραφο αυτή, θα εξετασθούν προβλήματα με την αντίστροφη φιλοσοφία δηλαδή με προσέγγιση από *κάτω προς τα επάνω* (bottom - up). Πιο συγκεκριμένα, η φιλοσοφία των προβλημάτων αυτών είναι από την αρχή να επιλύονται τα μικρότερα προβλήματα και σταδιακά να επιλύονται τα μεγαλύτερα ως σύνθεση των απλούστερων. Δηλαδή, ένας τυπικός αλγόριθμος αυτής της τεχνικής ξεκινά με τα επιμέρους μικρότερου μεγέθους υποπροβλήματα, που επιλύονται με τη χρήση κάποιου κανόνα ή τύπου. Μάλιστα συνηθέστατα τα προσωρινά αποτελέσματα αποθηκεύονται σε ένα πίνακα ώστε να χρησιμοποιηθούν αργότερα χωρίς να απαιτείται να υπολογισθούν για δεύτερη ή τρίτη φορά κ.λπ. Στη συνέχεια οι επιμέρους αυτές λύσεις συνθέτουν την κατάληξη της τελικής λύσης του αρχικού προβλήματος.

Αυτή η μέθοδος σχεδίασης αλγορίθμων χρησιμοποιείται κυρίως για την επίλυση προβλημάτων βελτιστοποίησης, δηλαδή όταν χρειάζεται να βρεθεί το ελάχιστο ή το μέγιστο κάποιου μεγέθους. Στην προσέγγιση αυτού του τύπου δεν υπάρχει η ίδια λογική επαναληπτικών εκτελέσεων που υπήρχε στη προηγούμενη προσέγγιση, αλλά υπάρχει ένας πίνακας από υπο-αποτελέσματα που καταλήγει στην τελική λύση του προβλήματος. Επομένως συμπερασματικά ακολουθείται η εξής προσέγγιση:

1. ξεκινά η λύση από το ελάχιστο στιγμιότυπο του προβλήματος,
2. υπολογίζονται σταδιακά αποτελέσματα όλο και μεγαλύτερων υποστιγμιότυπων,
3. καταλήγει στη σύνθεση.

Η τεχνική του δυναμικού προγραμματισμού παρουσιάζεται με τη βοήθεια ενός απλού σχετικά παραδείγματος.

Υπολογισμός δύναμης

Πολλές γλώσσες προγραμματισμού προσφέρουν έναν ιδιαίτερο τελεστή για τον υπολογισμό της δύναμης. Ωστόσο, υπάρχουν άλλες γλώσσες που δεν διαθέτουν τέτοιο τελεστή, οπότε ο προγραμματιστής πρέπει από μόνος του να κωδικοποιήσει αυτή τη λειτουργία και να την εντάξει σε δική του βιβλιοθήκη.

Ας θεωρήσουμε , λοιπόν, ότι πρέπει να υπολογίσουμε την ακέραια δύναμη ενός πραγματικού αριθμού, a^b. Μία πρώτη απλή προσέγγιση είναι ο επόμενος επαναληπτικός αλγόριθμος Δύναμη1, του οποίου η λειτουργία είναι ευνόητη.

```
Αλγόριθμος Δύναμη1
Δεδομένα  // a, b //
power ← 1
Για i από 1 μέχρι b
   power ← power * a
Τέλος_επανάληψης
Αποτελέσματα // power //
Τέλος Δύναμη1
```

Ο αλγόριθμος Δύναμη1 είναι μεν εύκολος στον προγραμματισμό και την κατανόηση του, αλλά δεν είναι ιδιαίτερα αποτελεσματικός, επειδή εκτελεί b πολλαπλασιασμούς για τον υπολογισμό της b-οστής δύναμης. Το ότι ο αλγόριθμος αυτός δεν είναι αποτελεσματικός, αποδεικνύεται με το απλό παράδειγμα ότι το a^{16} μπορεί να υπολογισθεί με τέσσερις πολλαπλασιασμούς ως εξής: $(((\alpha^2)^2)^2)^2$. Στη συνέχεια ακολουθεί ο νέος τρόπος επίλυσης του προβλήματος με χρήση της μεθόδου του δυναμικού προγραμματισμού, όπου η βασική ιδέα είναι η προσωρινή αποθήκευση κάποιων προηγούμενων δυνάμεων του αριθμού μέχρι τη σταδιακή κατάληξη στη δύναμη b. Συγκεκριμένα ο πίνακας *power* θα αποθηκεύει το αποτέλεσμα της ύψωσης στο τετράγωνο της προηγούμενης θέσης του πίνακα. Για παράδειγμα, αν θα υπολογιζόταν οι δυνάμεις του 2 (δηλαδή, έστω ότι a=2), τότε ο πίνακας στη θέση 0 θα είχε την τιμή 1, στη θέση 1 την τιμή 2, στη θέση 2 την τιμή 4, στη θέση 3 την τιμή 16 κοκ, όπως φαίνεται και στο επόμενο σχήμα.

	0	1	2	3	4	
	$2^0=1$	$2^1=2$	$2^2=4$	$2^4=16$	$2^8=256$	...
Δύναμη του 2	0-κή	1η	2η	4η	8η	...

***Σχ. 4.5** Αποθήκευση δυνάμεων του 2.*

Για τον υπολογισμό, λοιπόν, κάποιας δύναμης απαιτούνται οι κατάλληλες θέσεις του πίνακα. Λόγου χάριν, για την ανεύρεση του 2^7, αρκούν οι αποθηκευμένες δυνάμεις μέχρι και την 4η δύναμη του 2, διότι $2^7 = 2^4 * 2^2 * 2^1$. Ο αλγόριθμος που ακολουθεί παρουσιάζει τον υπολογισμό του a^b με τη νέα αυτή ευφυέστερη τεχνική, που βασίζεται στο λεγόμενο δυναμικό προγραμματισμό.

```
Αλγόριθμος Δύναμη2
Δεδομένα // a, b //
!Σχόλιο: αποθήκευση στοιχείων πίνακα
power[1] ← a
i ← 1
pow ← 1
Όσο pow < b επανάλαβε
    i ← i+1
    pow ← 2* pow
    power[i] ← power[i-1] * power[i-1]
Τέλος_επανάληψης
!Σχόλιο: ανεύρεση της δύναμης
used ← 0
result ← 1
Όσο used < b επανάλαβε
    Αν used + pow <= b τότε
        result ← result * power[i]
        used ← used + pow
    Τέλος_αν
    pow ← pow / 2
    i ← i - 1
Τέλος_επανάληψης
Αποτελέσματα // result //
Τέλος Δύναμη2
```

4.5 Άπληστη μέθοδος

Σε πολλά προβλήματα βελτιστοποίησης, όπου απαιτείται να βρεθεί η μέγιστη ή η ελάχιστη τιμή ενός μεγέθους, χρησιμοποιείται μία μέθοδος, που δεν βρίσκει πράγματι τη βέλτιστη λύση, αλλά σκοπός της είναι να την προσεγγίσει. Συνήθως, οι αλγόριθμοι αυτού του τύπου προχωρούν με βάση σταδιακές επιλογές που αφορούν στο βέλτιστο κάθε βήματος, χωρίς μέριμνα για το τελικό βέλτιστο. Έτσι η λειτουργία τους είναι παρόμοια με τις κινήσεις ενός παίκτη στο σκάκι που σκέφτεται και επιλέγει την επόμενη κίνηση χωρίς να τον απασχολούν οι επόμενες κινήσεις ούτε του αντιπάλου ούτε οι δικές του. Η πρώτη προσέγγιση στο παράδειγμα του ταχυδρόμου, το οποίο περιγράφεται στο Κεφάλαιο 4.1, ανήκει στην κατηγορία αυτή. Η μέθοδος αυτή που είναι γνωστή και ως "*άπληστη μέθοδος*" (greedy method), μπορεί να τυποποιηθεί σύμφωνα με τις εξής δύο παραδοχές:

- σε κάθε βήμα επιλογή της τρέχουσας βέλτιστης επιλογής
- επιβεβαίωση ότι αυτή η προσέγγιση εγγυάται τη συνολική βέλτιστη λύση (όταν ο αλγόριθμος δεν χρησιμοποιείται ως μια απλή ευριστική προσεγγιστική τακτική).

Η χρήση των αλγορίθμων αυτού του τύπου πρέπει να γίνεται με στόχο την ευφυέστερη επίλυση του προβλήματος, όπως παρουσιάζεται και στο παράδειγμα που ακολουθεί, όπου πράγματι βρίσκεται η βέλτιστη λύση.

Υπολογισμός νομισμάτων

Το νομισματικό μας σύστημα από 1/1/2002 διαθέτει κέρματα ονομαστικής αξίας των 1, 2, 5, 10, 20, 50 λεπτών, του 1 και των 2 ευρώ, καθώς και χαρτονομίσματα των 5, 10, 20, 50, 100, 200 και 500ευρώ. Το πρόβλημα είναι να βρεθεί αλγόριθμος με βάση τον οποίο, όταν δίδεται ένα ποσό, να βρίσκεται ο ελάχιστος αριθμός των απαιτούμενων κερμάτων και χαρτονομισμάτων. Για παράδειγμα για να σχηματιστεί το ποσό των 789€ απαιτούνται 1 χαρτονόμισμα των 500€, 1 των 200€, 1 των 50€, 1 των 20€, 1 των 10€, 1 των 5€ και 2 κέρματα των 2€, δηλαδή συνολικά 8 νομίσματα.

Το πρόβλημα αυτό μπορεί να επιλυθεί με "άπληστη" προσέγγιση, όπου κάθε στιγμή θα γίνεται επιλογή του νομίσματος που αντιστοιχεί στο μεγαλύτερο δυνατό ποσό. Τα βήματα επιλογής νομισμάτων περιγράφονται στον επόμενο αλγόριθμο, που παρουσιάζει μία απλοποιημένη έκδοση του προβλήματος.

```
Αλγόριθμος Νομίσματα
Δεδομένα  // C, n, poso //
find ← poso
coins ← 0
choice ← n
Όσο (choice>0) και (find > 0) επανάλαβε
    Αν C[choice]<= find τότε
        coins ← coins + 1
        find ← find - C[choice]
    αλλιώς
        choice ← choice - 1
    Τέλος_Αν
Τέλος_επανάληψης
Αποτελέσματα // coins //
Τέλος Νομίσματα
```

Στον αλγόριθμο αυτό ο πίνακας C διαθέτει όλες τις τιμές των n χρησιμοποιούμενων νομισμάτων από το μικρότερο προς το μεγαλύτερο. Ο αλγόριθμος υπολογίζει τον ελάχιστο αριθμό νομισμάτων για ένα συγκεκριμένο ποσό (μεταβλητή coins). Μπορεί να γίνει επέκτασή του, ώστε να έχει ως αποτέλεσμα τον αριθμό για κάθε είδος νομίσματος.

Ανακεφαλαίωση

Αρχικά δόθηκε ιδιαίτερη έμφαση στην έννοια και το ρόλο της ανάλυσης προβλημάτων πριν από την τελική επίλυσή τους με χρήση κάποιου αλγορίθμου. Τα μέρη της ανάλυσης ενός προβλήματος παρουσιάσθηκαν διεξοδικά και με χρήση παραδειγμάτων. Οι μέθοδοι σχεδίασης αλγορίθμων τυποποιήθηκαν και αναλύθηκαν με χρήση δύο βασικών προσεγγίσεων: της μεθόδου Διαίρει και Βασίλευε και του Δυναμικού Προγραμματισμού, που αποτελούν δύο συμπληρωματικές τεχνικές με αντίθετη φιλοσοφία. Ειδικότερα για την τεχνική Διαίρει και Βασίλευε παρουσιάσθηκε ο αναδρομικός αλγόριθμος της δυαδικής αναζήτησης. Ο αλγόριθμος αυτός είναι τυπικό παράδειγμα μιας κλάσης προβλημάτων που έχουν επιλυθεί με τέτοιου είδους προσέγγιση. Σε σχέση με τη μέθοδο του Δυναμικού Προγραμματισμού δόθηκε η λογική της λειτουργίας της και παρουσιάσθηκαν αλγόριθμοι και παραδείγματα για την επίλυση απλών προβλημάτων όπως η ύψωση σε δύναμη. Επίσης, παρουσιάσθηκε η φιλοσοφία της λεγόμενης Άπληστης μεθόδου και δόθηκε ένα πιο σύνθετο παράδειγμα, όπως είναι η επιστροφή από ρέστα με το μικρότερο αριθμό κερμάτων και χαρτονομισμάτων.

Λέξεις κλειδιά

Διαίρει και βασίλευε, Δυναμικός προγραμματισμός, Άπληστη μέθοδος, Δυαδική αναζήτηση, Μέθοδος διχοτόμησης.

Ερωτήσεις - Θέματα για συζήτηση

1. Τι περιλαμβάνει η ανάλυση ενός προβλήματος σε ένα σύγχρονο υπολογιστικό περιβάλλον;
2. Ποιοί είναι οι λόγοι για τους οποίους οι μέθοδοι ανάλυσης και επίλυσης των προβλημάτων παρουσιάζουν ιδιαίτερο ενδιαφέρον;
3. Ποιά είναι τα δύο βασικά είδη προσέγγισης της επίλυσης ενός προβλήματος;

4. Να περιγραφεί με βήματα η μέθοδος της προσέγγισης Διαίρει και Βασίλευε για τη σχεδίαση αλγορίθμων.
5. Να περιγραφεί με βήματα η μέθοδος της προσέγγισης του Δυναμικού Προγραμματισμού για τη σχεδίαση αλγορίθμων.
6. Να περιγραφεί η μέθοδος της δυαδικής αναζήτησης και να δοθεί ένα παράδειγμα επίλυσης προβλήματος με τη μέθοδο αυτή.
7. Να δοθεί αλγόριθμος για την ανεύρεση του παραγοντικού ακεραίου με χρήση της από προσέγγισης, όπου τα προσωρινά αποτελέσματα αποθηκεύονται σε έναν πίνακα.
8. Να δοθεί αλγόριθμος για την ανεύρεση της δύναμης πραγματικού σε αρνητικό ακέραιο εκθέτη με Δυναμικό Προγραμματισμό.

Βιβλιογραφία

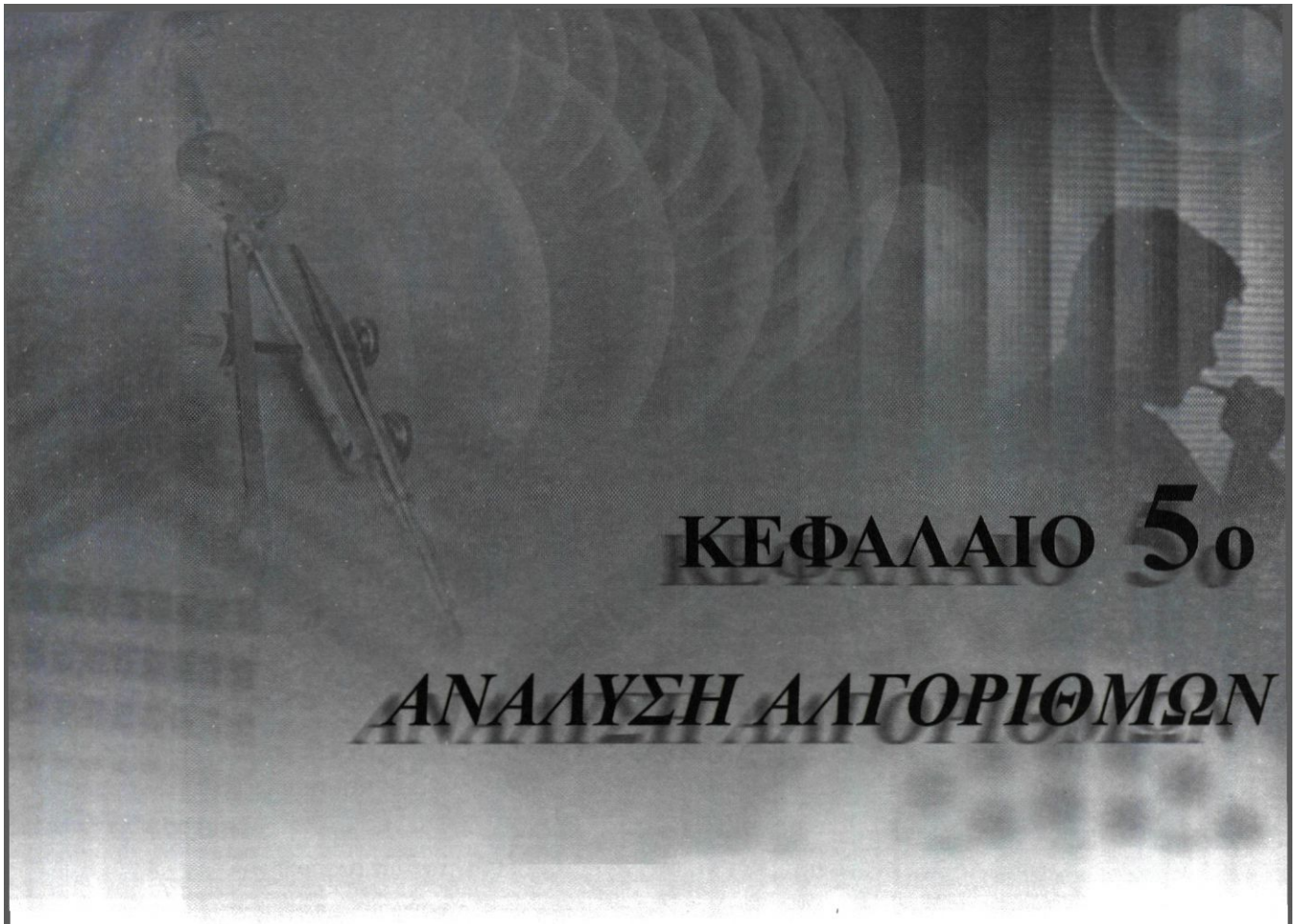
1. Φώτω Αφράτη και Γιώργος Παπαγεωργίου: Αλγόριθμοι – Μέθοδοι Σχεδίασης και Ανάλυση Πολυπλοκότητας, Εκδόσεις Συμμετρία, Αθήνα, 1993.
2. Χρήστος Κοίλιας: Δομές Δεδομένων και Οργάνωση Αρχείων. Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1993.
3. Μανόλης Λουκάκης: Δομές Δεδομένων και Αλγόριθμοι, Έκδοσεις Θεραπευτικής Κοινότητας Ιθάκη, Θεσσαλονίκη, 1988.
4. Ιωάννης Μανωλόπουλος: Δομές Δεδομένων – μία Προσέγγιση με Pascal, Εκδόσεις Art of Text, Θεσσαλονίκη, 1998.
5. Ιωάννης Μανωλόπουλος, Μαθήματα Θεωρίας Γράφων, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1996.
6. Νίκος Μισυρλής, Δομές Δεδομένων, Αθήνα, 1995.
7. Niklaus Wirth: Αλγόριθμοι και Δομές Δεδομένων, Κλειδάριθμος, Αθήνα, 1990.
8. G. Brassard and P.Bratley: *"Fundamentals of Algorithms"*, Prentice Hall, 1996.
9. T. Cormen, C. Leiserson and R. Rivest: *"Introduction to Algorithms"* MIT Press, 1990.
10. E. Horowitz, S. Sahni and S. Rajasekaran: *"Computer Algorithms"*, Computer Science Press, 1998.

11. I. Oliver: "Programming Classics: Implementing the World's Best Algorithms", Prentice Hall, 1993.

Διευθύνσεις Διαδικτύου

http://www.cs.pitt.edu/~kirk/algorithmcourses/index.html

Αποτελεί κόμβο με πλούσιο πληροφορικό υλικό υψηλού ακαδημαϊκού επίπέδου. Περιέχει και στοιχεία βασικού επιπέδου.



ΚΕΦΑΛΑΙΟ 5ο

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

ΠΕΡΙΕΧΟΜΕΝΑ

- Επίδοση αλγορίθμων
- Ορθότητα αλγορίθμων
- Πολυπλοκότητα αλγορίθμων
- Είδη αλγορίθμων

Εισαγωγή

Η καταγραφή των μεγεθών που επηρεάζουν την επίδοση ενός αλγορίθμου είναι μία σημαντική ενέργεια για την κατανόηση της αποδοτικότητας ενός αλγορίθμου. Για το σκοπό αυτό πρέπει να βρεθεί ποιά είναι η βασική λειτουργία και ποιά η δομή του αλγορίθμου, όπου δαπανώνται κυρίως οι υπολογιστικοί πόροι (δηλαδή, ο χρόνος). Η πολυπλοκότητα των αλγορίθμων, λοιπόν, περιγράφεται αναλυτικά και τυποποιούνται οι ορισμοί για τις κατηγορίες (τάξεις) πολυπλοκότητας των περισσοτέρων ειδών αλγορίθμων. Για μερικούς από τους αλγορίθμους, που γνωρίσαμε στα προηγούμενα κεφάλαια, γίνεται ανάλυση και εκφράζεται η πολυπλοκότητά τους και τέλος, γίνεται μια παρουσίαση των ειδών των αλγορίθμων.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι οι μαθητές :

- να περιγράφουν τις τεχνικές ανάλυσης των αλγορίθμων,
- να μετρούν τη επίδοση των αλγορίθμων,
- να αντιπαραβάλλουν αλγόριθμους με βάση τυποποιημένους κανόνες επιλογής,
- να μπορούν να ελέγχουν την ορθότητα των αλγορίθμων,
- να διατυπώνουν την έννοια της πολυπλοκότητας των αλγορίθμων,
- να συνοψίζουν τα κυριότερα είδη αλγορίθμων.

Προερωτήσεις

✓ Έχεις αναρωτηθεί πόσο γρήγορα μπορεί να γίνει η αναζήτηση ενός στοιχείου ή η ταξινόμηση κάποιων αριθμών;

✓ Ξέρεις αν υπάρχουν προβλήματα άλυτα;

5.1 Επίδοση αλγορίθμων

Στα προηγούμενα κεφάλαια του βιβλίου παρουσιάσθηκαν πολλοί αλγόριθμοι για την επίλυση ενός προβλήματος, όπως για παράδειγμα σχετικά με την αναζήτηση και την ταξινόμηση. Στο σημείο αυτό θα παρουσιάσουμε έναν τρόπο εκτίμησης της *επίδοσης* (performance) ή της *αποδοτικότητας* (efficiency) των αλγορίθμων.

Για την κατανόηση της επίδοσης ενός αλγορίθμου χρειάζεται να απαντηθεί ένα σύνολο ερωτημάτων. Τα πρωταρχικά ερωτήματα που προκύπτουν είναι:

1. πώς υπολογίζεται ο χρόνος εκτέλεσης ενός αλγορίθμου;
2. πώς μπορούν να συγκριθούν μεταξύ τους οι διάφοροι αλγόριθμοι;
3. πώς μπορεί να γνωρίζει κανείς, αν ένας αλγόριθμος είναι "βέλτιστος";

Η απάντηση στα ερωτήματα αυτά προαπαιτεί την καταγραφή ουσιωδών πληροφοριών για το πρόβλημα. Οι πληροφορίες αυτές αφορούν κυρίως στην αναγνώριση της χειρότερης περίπτωσης του αλγορίθμου και στην αποτύπωση του μεγέθους του προβλήματος με βάση το πλήθος των δεδομένων.

5.1.1 Χειρότερη περίπτωση ενός αλγορίθμου

Η χειρότερη περίπτωση ενός αλγορίθμου αφορά στο μέγιστο κόστος εκτέλεσης του αλγορίθμου, κόστος που μετράται σε υπολογιστικούς πόρους. Το κόστος αυτό πολλές φορές κρίνει την επιλογή και το σχεδιασμό ενός αλγορίθμου. Για να εκφρασθεί αυτή η χειρότερη περίπτωση χρειάζεται κάποιο μέγεθος σύγκρισης και αναφοράς που να χαρακτηρίζει τον αλγόριθμο. Η πλέον συνηθισμένη πρακτική είναι η μέτρηση του αριθμού των *βασικών πράξεων* που θα πρέπει να εκτελέσει ο αλγόριθμος στη χειρότερη περίπτωση. Για παράδειγμα, μία βασική πράξη μπορεί να είναι :

- ✓ ανάθεση τιμής,
- ✓ σύγκριση μεταξύ δύο μεταβλητών, ή
- ✓ οποιαδήποτε αριθμητική πράξη μεταξύ δύο μεταβλητών.

Η χειρότερη περίπτωση αντιπροσωπεύει τις τιμές εκείνες, που όταν δίνονται ως είσοδος στον αλγόριθμο, οδηγούν στην εκτέλεση μέγιστου αριθμού βασικών πράξεων.

Παράδειγμα: έστω ότι δίνεται ο επόμενος απλός αλγόριθμος :

```
Αλγόριθμος Παράδειγμα1
n ← 10
Αρχή_επανάληψης
    Διάβασε m
    n ← n - 1
Μέχρις_ότου (m=0) ή (n=0)
Εκτύπωσε m
Τέλος Παράδειγμα1
```

Είναι προφανές ότι η χειρότερη περίπτωση για αυτόν τον αλγόριθμο προκύπτει όταν γίνουν 10 επαναλήψεις (δηλαδή μέχρι να ισχύει το n=0).

5.1.2 Μέγεθος εισόδου ενός αλγορίθμου

Πρέπει να υπάρχει κάποια ή κάποιες μεταβλητές που να εκφράζουν το *μέγεθος* (size) του αλγορίθμου. Οι μεταβλητές αυτές απεικονίζουν τους διαφορετικούς συνδυασμούς τιμών που κρίνουν τη συμπεριφορά ενός αλγορίθμου και δίνονται ως δεδομένα στον αλγόριθμο. Γενικά, τα δεδομένα συνιστούν το μέγεθος της εισόδου ενός αλγορίθμου. Για παράδειγμα, στον αλγόριθμο του προηγούμενου παραδείγματος, το μέγεθος του αλγορίθμου μπορεί να εκφρασθεί από τη μεταβλητή n, που στην ουσία εκφράζει το πλήθος των επαναλήψεων του βασικού βρόχου του αλγορίθμου.

Ο πίνακας 5.1 περιλαμβάνει παραδείγματα από συνήθεις κατηγορίες προβλημάτων, με δήλωση του μεγέθους που βασικά εκφράζει την είσοδό τους και των αντίστοιχων βασικών πράξεων:

Πιν. 5.1. *Μέγεθος εισόδου και βασική πράξη αλγορίθμων*

Αλγόριθμος	Μέγεθος εισόδου αλγορίθμου (n)	Βασική Πράξη
ΤΑΞΙΝΟΜΗΣΗ	το πλήθος των αντικειμένων που θα ταξινομηθούν	σύγκριση
ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ	το πλήθος των ψηφίων των αριθμών που θα πολλαπλασιασθούν	αριθμητικές πράξεις
ΑΝΑΖΗΤΗΣΗ	το πλήθος των στοιχείων του πίνακα	σύγκριση

5.1.3 Χρόνος εκτέλεσης προγράμματος ενός αλγορίθμου

Έστω ότι έχουμε τον εξής αλγόριθμο:

```
Αλγόριθμος Παράδειγμα2
x ← 123
y ← 234
Για i από 0 μέχρι 4
    Εκτύπωσε i
    z ← x * y
Τέλος_επανάληψης
Εκτύπωσε x
Εκτύπωσε y
Εκτύπωσε z
Τέλος Παράδειγμα2
```

Θα υπολογισθεί η επίδοσή του με βάση τον αριθμό των πράξεων που θα εκτελεσθούν. Με δεδομένο ότι ο βρόχος του προγράμματος θα εκτελεσθεί 5 φορές, προκύπτει η παρακάτω ανάλυση :

Εντολή αλγορίθμου	Αριθμός πράξεων
ανάθεση τιμών στα x και y	2
Βρόχος επανάληψης	
αρχική τιμή i	1
έλεγχος i	6
αύξηση i	5
εκτύπωση i	5
υπολογισμός z (2X5)	10
Εκτύπωση x, y, z	3
ΣΥΝΟΛΟ	***32***

Οι βρόχοι επανάληψης αποτελούν το κρίσιμο σημείο για τον χαρακτηρισμό της επίδοσης ενός αλγορίθμου Έτσι αν ο αλγόριθμος αυτός γενικευθεί ώστε ο βρόχος να εκτελεσθεί n φορές, ο χρόνος εκτέλεσης θα εξαρτάται από το μέγεθος του n. Ο πίνακας 5.2 παρουσιάζει τους χρόνους εκτέλεσης του αλγορίθμου αυτού για διαφορετικά μεγέθη του n, θεωρώντας ότι ο χρόνος είναι ο αριθμός των πράξεων σε μικρο-δευτερόλεπτα :

Πιν. 5.2. *Χρόνοι εκτέλεσης αλγορίθμου ανάλογα με το μέγεθος*

μέγεθος n	Χρόνος εκτέλεσης
5	32 μικρο-δευτερόλεπτα
10	57 μικρο-δευτερόλεπτα
100	507 μικρο-δευτερόλεπτα
1.000.000	5 δευτερόλεπτα (περίπου)

5.1.4 Αποδοτικότητα αλγορίθμων

Αν η επίλυση ενός προβλήματος επιτυγχάνεται με τη χρήση δύο ή περισσοτέρων αλγορίθμων, χρειάζεται να γίνει η επιλογή του καταλληλότερου με βάση την αποδοτικότητά τους. Έτσι, αν ο αλγόριθμος Β έχει το ίδιο αποτέλεσμα με τον αλγόριθμο Α, αλλά δίνει τα αποτελέσματα σε λιγότερο χρόνο, τότε είναι αποδοτικότερος του Α. Με παρόμοιο τρόπο όταν ο αλγόριθμος Β έχει το ίδιο αποτέλεσμα με έναν αλγόριθμο Α, αλλά έχει τα αποτελέσματα με χρήση λιγότερης μνήμης, τότε είναι αποδοτικότερος του Α. Βέβαια όταν συγκρίνονται δύο αλγόριθμοι, θα πρέπει να συγκρίνονται με χρήση των ίδιων δεδομένων και κάτω από τις ίδιες συνθήκες. Γενικά, ο χρόνος εκτέλεσης κάθε αλγορίθμου εξαρτάται από ένα σύνολο παραγόντων που μπορούν να συνοψισθούν στους εξής:

- Τύπος ηλεκτρονικού υπολογιστή που θα εκτελέσει το πρόγραμμα του αλγορίθμου,
- Γλώσσα προγραμματισμού που θα χρησιμοποιηθεί,
- Δομή προγράμματος και δομές δεδομένων που χρησιμοποιούνται,
- Χρόνος για πρόσβαση στο δίσκο και στις ενέργειες εισόδου-εξόδου,
- Είδος συστήματος, ενός χρήστη ή πολλών χρηστών.

Επομένως, για να έχει έννοια κάθε σύγκριση μεταξύ δύο προγραμμάτων αλγορίθμων, θα πρέπει να ικανοποιούνται οι παρακάτω προϋποθέσεις:

- ✓ και τα δύο προγράμματα να έχουν συνταχθεί στην ίδια γλώσσα προγραμματισμού,
- ✓ να έχει χρησιμοποιηθεί ο ίδιος μεταφραστής της γλώσσας προγραμματισμού,
- ✓ να χρησιμοποιείται η ίδια υπολογιστική πλατφόρμα,

✓ ακριβώς τα ίδια δεδομένα να αποτελούν είσοδο κατά τον έλεγχο των δύο αλγορίθμων.

5.2 Ορθότητα αλγορίθμων

Η επίλυση ενός προβλήματος με τη χρήση κάποιου αλγορίθμου, έχει μεγαλύτερη ισχύ όταν υπάρχει κάποια τυποποιημένη ένδειξη ή απόδειξη για την ορθότητα του προτεινόμενου αλγορίθμου. Είναι ιδιαίτερα σημαντικό να επικυρωθεί η ορθότητα και εγκυρότητα ενός αλγορίθμου με χρήση κάποιων απλών μαθηματικών τύπων ή μεθόδων. Οι τεχνικές απόδειξης της ορθότητας ενός αλγορίθμου συνδέονται άμεσα με τον τρόπο της αρχικής σχεδίασης και ανάλυσης του συγκεκριμένου αλγορίθμου. Είναι λάθος να επιχειρείται ο έλεγχος των σφαλμάτων και της ορθότητας ενός αλγορίθμου μετά τη σχεδίαση και την τυποποίησή του. Είναι, δηλαδή, απαραίτητο να συνοδεύεται το στάδιο της ανάλυσης και της σχεδίασης του αλγορίθμου από τον αντίστοιχο έλεγχο της ορθότητάς του. Στη συνέχεια θα παρουσιασθούν οι περιορισμοί, οι τεχνικές και οι μέθοδοι προσέγγισης των αποδείξεων της ορθότητας των αλγορίθμων.

Περιορισμοί ελέγχου ορθότητας ενός αλγορίθμου

Η πρόταση ενός αλγορίθμου καταλήγει στη σύνταξη κώδικα για την επίλυσή του στον υπολογιστή. Αν ένας προγραμματιστής έχει υλοποιήσει κώδικα για την υλοποίηση ενός αλγορίθμου, μπορεί να ελέγξει το πρόγραμμα "τρέχοντας" τον κώδικα μία, δύο, τρεις κ.λπ. φορές. Έστω ότι σε όλες τις περιπτώσεις καταλήγει σε σωστά αποτελέσματα. Είναι όμως βέβαιο ότι, ο αλγόριθμος του προγράμματος δίνει πάντοτε τη σωστή λύση; Η απάντηση είναι όχι, διότι υπάρχει πιθανότητα κάποια φορά να καταλήξει σε λάθος αποτέλεσμα λόγω κάποιου λάθους στον αλγόριθμο. Το παράδειγμα που ακολουθεί παρουσιάζει μία τέτοια περίπτωση :

Παράδειγμα: Να βρεθεί ο μικρότερος αριθμός από το σύνολο των θετικών αριθμών που βρίσκονται αποθηκευμένοι σε ένα πίνακα 10 θέσεων.

Στη συνέχεια δίνεται ο σχετικός αλγόριθμος για την εύρεση του μικρότερου στοιχείου σε πίνακα:

```
Αλγόριθμος Παράδειγμα3
Δεδομένα  // a //
low ← a[1]
i ← 2
Όσο i < 10 επανάλαβε
    Αν a[i] < low τότε low ← a[i]
    i ← i+1
Τέλος_επανάληψης
Αποτελέσματα // low //
Τέλος Παράδειγμα3
```

Από τον ψευδοκώδικα αυτό μπορεί να θεωρήσει κανείς ότι, ο αλγόριθμος είναι σωστός και παράγει το σωστό αποτέλεσμα. Η ορθότητα του αλγορίθμου θα ελεγχθεί με ένα σύνολο από πίνακες 10 θέσεων, που έχουν τυχαίους αριθμούς. Έστω ότι προκύπτουν οι επόμενοι πίνακες από 3 διαφορετικές εκτελέσεις του προγράμματος του αλγορίθμου :

11	3	2	56	32	69	81	90	222	2

2	444	1	65	51	51	99	98	1	1

90	11	333	38	224	61	73	80	7	59

Η εφαρμογή του αλγορίθμου σε κάθε έναν από τους πίνακες αυτούς, θα δώσει το σωστό αποτέλεσμα για το μικρότερο στοιχείο του πίνακα. Αν όμως δοθεί ο πίνακας με τα εξής δεδομένα:

6	8	3	4	3	7	8	9	5	2

τότε ο αλγόριθμος θα δώσει ως αποτέλεσμα το 3, ενώ το μικρότερο στοιχείο είναι το 2. Άρα είναι σαφές ότι, υπάρχει κάποιο σφάλμα στον αλγόριθμο, που εδώ παρουσιάστηκε με την τέταρτη εκτέλεση του αλγορίθμου. Θα μπορούσε να είχε παρουσιασθεί το σφάλμα πολύ αργότερα, δηλαδή μετά από μεγάλο αριθμό επαναληπτικών εκτελέσεων του αλγορίθμου.

Γενικότερα, λοιπόν, δεν υπάρχει καμμία εγγύηση ή επιβεβαίωση του αριθμού των επαναλήψεων εκτέλεσης ενός αλγορίθμου, μέχρι να βρεθεί κάποιο πιθανό σφάλμα του. Το σφάλμα στον προηγούμενο αλγόριθμο είναι η συνθήκη (i < 10), που δεν ελέγχει ποτέ την τελευταία θέση του πίνακα.

Έτσι αν το μικρότερο στοιχείο βρίσκεται στην τελευταία θέση του πίνακα, τότε ο αλγόριθμος θα επιστρέψει λανθασμένη τιμή. Βέβαια, υπάρχει περίπτωση να επιστρέψει το σωστό αποτέλεσμα, μόνο αν το μικρότερο στοιχείο που είναι στην τελευταία θέση, έχει εμφανισθεί ήδη και σε κάποια άλλη μικρότερη θέση του πίνακα.

Για τον υπολογισμό της πιθανοτήτας να υπάρξει λάθος κατά την εκτέλεση του αλγορίθμου, παρατηρούμε ότι η πιθανότητα αυτή προκύπτει, όταν το μικρότερο στοιχείο βρίσκεται στην τελευταία θέση του πίνακα. Άρα η πιθανότητα είναι 1/10, αφού υπάρχουν 10 θέσεις στον πίνακα. Έτσι η πιθανότητα να μη βρεθεί το λάθος με μία προσπάθεια είναι 0,9. Αν γίνουν δύο προσπάθειες, λόγω της τυχαιότητας των στοιχείων του πίνακα η πιθανότητα αυτή γίνεται 0,9*0,9=0,81. Όσο επαναλαμβάνεται η εκτέλεση του αλγορίθμου για κάποια επιπλέον τυχαία ακολουθία δεδομένων οι πιθανότητες διαμορφώνονται ως εξής :

Αριθμός Εκτελέσεων Αλγορίθμου	Πιθανότητα Ανεύρεσης Σφάλματος	Πιθανότητα Μη-Ανεύρεσης Σφάλματος
1	0,1	0,9
2	0,19	0,81
3	0,271	0,729
4	0,344	0,656
5	0,410	0,590
6	0,469	0,531
7	0,522	0,478

Από αυτόν τον πίνακα πιθανοτήτων, είναι φανερό ότι χρειάζονται τουλάχιστον 7 προσπάθειες εκτέλεσης του αλγορίθμου για να υπάρχει πιθανότητα της τάξης του 0,5 να βρεθεί ένα σφάλμα. Η παραπάνω καταγραφή των πιθανοτήτων αποτελεί έναν τρόπο εκτίμησης για τον εντοπισμό λαθών ενός αλγορίθμου. Ωστόσο, δεν μπορεί να υπάρξει απόλυτη ασφάλεια στην εκτίμηση των πιθανών σφαλμάτων ενός αλγορίθμου. Γενικότερα δεν μπορεί να υπάρξει απόλυτη επιβεβαίωση της εγκυρότητας ενός αλγορίθμου με χρήση κάποιας πιθανολογικής εκτίμησης των πιθανών σφαλμάτων του.

Η απόδειξη της ορθότητας ενός αλγορίθμου θα πρέπει να περιλαμβάνει τις εξής δύο συνθήκες :

- απόδειξη ότι σε κάθε περίπτωση ο τερματισμός της εκτέλεσης του αλγορίθμου οδηγεί σε αποδεκτά αποτελέσματα, και
- απόδειξη ότι θα υπάρξει τερματισμός της εκτέλεσης του αλγορίθμου

Γενικά, η απόδειξη της ορθότητας ενός αλγορίθμου δεν είναι εύκολη υπόθεση. Στο σημείο αυτό δίνουμε μόνο ένα απλό παράδειγμα.

Έστω ότι δίνονται δύο μεταβλητές a και b με τις αντίστοιχες τιμές τους και ότι πρέπει να παρουσιασθεί ένας αλγόριθμος ανταλλαγής των τιμών τους. Αν υπάρχει εκ των προτέρων πρόβλεψη για την ορθότητα ενός αλγορίθμου ανταλλαγής των τιμών δύο μεταβλητών, τότε θα πρέπει να ακολουθηθεί κάποια τυποποίηση και στην ονοματολογία των μεταβλητών. Αυτό είναι απαραίτητο, γιατί είναι σαφές ότι η ανταλλαγή των μεταβλητών όπως: $a \leftarrow b$ και $b \leftarrow a$, δεν οδηγεί πράγματι στην ανταλλαγή των τιμών τους. Επομένως, οι αρχικές τιμές των μεταβλητών πρέπει να αποθηκευθούν σε κάποιες προσωρινές μεταβλητές a_0 και b_0 αντίστοιχα. Έτσι, ο στόχος του προβλήματος είναι να ισχύει: $a=b_0$ και $b=a_0$. Με αυτήν την ονοματολογία το πρόβλημα αντιμετωπίζεται στη γενική του μορφή. Ωστόσο, η προηγούμενη τυποποίηση είναι απλά ενδεικτική, γιατί τελικά επαρκεί μία επιπλέον μεταβλητή που θα λειτουργήσει ως ενδιάμεσος χώρος για την ανταλλαγή των τιμών των μεταβλητών. Ο προτεινόμενος αλγόριθμος λοιπόν είναι:

$t \leftarrow a$, $a \leftarrow b$ και $b \leftarrow t$

Σύμφωνα με τα προηγούμενα, είναι προφανές ότι, ο αλγόριθμος που στηρίζεται στην τεχνική αυτή είναι ορθός.

5.3 Πολυπλοκότητα αλγορίθμων

Ο απλούστερος τρόπος μέτρησης της επίδοσης ενός αλγορίθμου είναι ο *εμπειρικός* (empirical) ή αλλιώς ο λεγόμενος *εκ των υστέρων* (a posteriori). Δηλαδή, ο αλγόριθμος υλοποιείται και εφαρμόζεται σε ένα σύνολο δεδομένων, ώστε να υπολογισθεί ο απαιτούμενος *χρόνος επεξεργασίας* (processing time) και η *χωρητικότητα μνήμης* (memory space). Ο τρόπος όμως αυτός παρουσιάζει δύο κύρια μειονεκτήματα.

- με τη μέθοδο αυτή είναι δύσκολο να προβλεφθεί η συμπεριφορά του αλγορίθμου για κάποιο άλλο σύνολο δεδομένων.
- ο χρόνος επεξεργασίας εξαρτάται από το υλικό, τη γλώσσα προγραμματισμού και το μεταφραστή και προ πάντων από τη δεινότητα του προγραμματιστή.

Έτσι μπορεί να συναχθούν λανθασμένες εκτιμήσεις για την επίδοση του αλγορίθμου.

Ο δεύτερος τρόπος εκτίμησης της επίδοσης ενός αλγορίθμου είναι ο *θεωρητικός* (theoretical) ή αλλιώς ο λεγόμενος *εκ των προτέρων* (a priori). Για το σκοπό αυτό εισάγεται μία μεταβλητή n, που εκφράζει το μέγεθος (size) του προβλήματος, ώστε η μέτρηση της αποδοτικότητας του αλγόριθμου να ισχύει για οποιοδήποτε σύνολο δεδομένων και ανεξάρτητα από υποκειμενικούς παράγοντες, όπως αυτοί που αναφέρθηκαν. Η σημασία της μεταβλητής αυτής εξαρτάται από το πρόβλημα, που πρόκειται να επιλυθεί. Για παράδειγμα, αν το πρόβλημα είναι η ταξινόμηση k στοιχείων τότε n=k. Στη συνέχεια ο χρόνος επεξεργασίας και ο απαιτούμενος χώρος μνήμης εκτιμώνται με τη βοήθεια μίας συνάρτησης f(n) που εκφράζει τη *χρονική πολυπλοκότητα* (time complexity) ή την *πολυπλοκότητα χώρου* (space complexity). Σε πολλές περιπτώσεις όμως δεν ενδιαφέρουν οι επακριβείς τιμές αλλά μόνο η γενική συμπεριφορά των αλγορίθμων, δηλαδή η τάξη του αλγορίθμου. Για το λόγο αυτό εισάγεται ο λεγόμενος *συμβολισμός Ο* (O-notation), όπου το Ο είναι το αρχικό γράμμα της αγγλικής λέξης order και διαβάζεται *"όμικρον κεφαλαίο"*. Ακολουθεί όμως ένας τυπικός ορισμός.

Ορισμός: Αν η πολυπλοκότητα ενός αλγορίθμου είναι f(n), τότε λέγεται ότι είναι τάξης O(g(n)), αν υπάρχουν δύο θετικοί ακέραιοι c και n_0, έτσι ώστε για κάθε $n \geq n_0$ να ισχύει:

$$|f(n)| \leq c|g(n)|$$

Παράδειγμα. Έστω ότι η πολυπλοκότητα ενός αλγορίθμου δίνεται από το πολυώνυμο $f(n)=2n^3+5n^2-4n+3$. Ο πιο σημαντικός όρος του πολυωνύμου είναι η τρίτη δύναμη, αρκεί να σκεφτούμε ότι καθώς το x αυξάνεται (και τείνει στο άπειρο) ο όρος αυτός έχει μεγαλύτερο «ειδικό βάρος» και συνεχώς καθίσταται σημαντικότερος, ενώ η σημασία των υπολοίπων όρων σταδιακά μειώνεται. Επιπλέον, αγνοείται ο συντελεστής 2 της τρίτης δύναμης και έτσι προκύπτει ότι $g(n)=n^3$. Επομένως, τελικά η πολυπλοκότητα του αλγορίθμου είναι $O(n^3)$. Πλέον, η έκφραση αυτή εκφράζει την ποιοτική και όχι την ποσοτική συμπεριφορά του αλγορίθμου. Κατά τον ίδιο τρόπο αν δοθεί η έκφραση $f(n)=5\cdot 2^n+4n^2-4\log n$, τότε προκύπτει ότι $g(n)=2^n$, καθώς αγνοούνται οι μη σημαντικοί όροι του f(n) και ο συντελεστής 5 του όρου 2^n. Στο τελευταίο παράδειγμα είναι περισσότερο σαφές, γιατί αγνοούμε το δεύτερο και τρίτο όρο του f(n), αν απλά δώσουμε τιμές 1, 2, 3 κλπ στη μεταβλητή n και υπολογίσουμε το αποτέλεσμα του κάθε όρου της έκφρασης. Θα παρατηρήσουμε ότι πολύ σύντομα (λόγου χάριν, για n=10), το τελικό αποτέλεσμα της έκφρασης κατά βάση προέρχεται από τη δύναμη 2^n.

Λέγεται ότι ο συμβολισμός Ο δίνει ένα άνω φράγμα για την πολυπλοκό-

τητα ενός αλγορίθμου, δηλαδή δίνει ένα μέτρο, που ποτέ δεν πρόκειται να ξεπεράσει ο αλγόριθμος προς τα επάνω. Ωστόσο, στη βιβλιογραφία υπάρχουν και άλλοι συμβολισμοί, αλλά δεν θα μας απασχολήσουν στα πλαίσια αυτού του βιβλίου.

Σχεδόν οι περισσότεροι αλγόριθμοι πρακτικού ενδιαφέροντος έχουν χρονική πολυπλοκότητα, που ανήκει σε μία από τις επόμενες κατηγορίες:

- O(1). Κάθε εντολή του προγράμματος εκτελείται μία φορά ή το πολύ μερικές μόνο φορές. Στην περίπτωση αυτή λέγεται ότι ο αλγόριθμος είναι *σταθερής πολυπλοκότητας*.
- O(logn). Ο αλγόριθμος είναι *λογαριθμικής πολυπλοκότητας*. Ας σημειωθεί ότι με "log" θα συμβολίζεται ο δυαδικός λογάριθμος, ενώ με "ln" θα συμβολίζεται ο φυσικός λογάριθμος. Συνήθως, οι λογάριθμοι που χρησιμοποιούνται στο βιβλίο αυτό είναι δυαδικοί.
- O(n). Η πολυπλοκότητα λέγεται *γραμμική*. Αυτή είναι η καλύτερη επίδοση για έναν αλγόριθμο που πρέπει να εξετάσει ή να δώσει στην έξοδο n στοιχεία.
- O(n logn). Διαβάζεται όπως ακριβώς γράφεται (n logn), δηλαδή χωρίς να χρησιμοποιείται κάποιο επίθετο (όπως για παράδειγμα γραμμολογαριθμική). Στην κατηγορία αυτή ανήκει μία πολύ σπουδαία οικογένεια αλγορίθμων ταξινόμησης.
- $O(n^2)$. *Τετραγωνική πολυπλοκότητα*. Πρέπει να χρησιμοποιείται μόνο για προβλήματα μικρού μεγέθους.
- $O(n^3)$. *Κυβική πολυπλοκότητα*. Και αυτοί οι αλγόριθμοι πρέπει να χρησιμοποιούνται μόνο για προβλήματα μικρού μεγέθους.
- $O(2^n)$. Σπάνια στην πράξη χρησιμοποιούνται αλγόριθμοι *εκθετικής πολυπλοκότητας*.

Στον πίνακα 5.3 υπολογίζεται ο χρόνος που απαιτείται από αλγορίθμους διαφόρων πολυωνυμικών και εκθετικών πολυπλοκοτήτων ως συνάρτηση του μεγέθους του προβλήματος (n). Βέβαια υποτίθεται ότι κάθε στοιχειώδης πράξη απαιτεί ένα μικροδευτερόλεπτο. Έτσι αν για έναν αλγόριθμο τάξης $O(n^3)$ διπλασιασθεί το μέγεθος του προβλήματος, τότε απαιτείται οκταπλάσιος (2^3) χρόνος για να περατωθεί ο αλγόριθμος. Επίσης διαπιστώνεται αμέσως ότι, οι αλγόριθμοι εκθετικής πολυπλοκότητας δεν είναι πρακτικής χρησιμότητας ακόμη και για μικρό αριθμό δεδομένων.

Πιν. 5.3. *Χρόνοι εκτέλεσης αλγορίθμων ανάλογα με την πολυπλοκότητα και το μέγεθος*

Πολυπλοκότητα αλγορίθμου	Μέγεθος προβλήματος		
	n=20	n=40	n=60
O(n)	0.00002 δεύτερα	0.00004 δεύτερα	0.00006 δεύτερα
$O(n^2)$	0.0004 δεύτερα	0.0016 δεύτερα	0.0036 δεύτερα
$O(n^3)$	0.008 δεύτερα	0.064 δεύτερα	216 δεύτερα
$O(2^n)$	1.0 δεύτερο	2.7 ημέρες	366 αιώνες
O(n!)	771 αιώνες	$3\ 10^{32}$ αιώνες	$3\ 10^{66}$ αιώνες

Η ανάλυση αλγορίθμων είναι ιδιαίτερα σημαντικό κεφάλαιο για όσους μελετούν βαθύτερα την επίδοση αλγορίθμων. Σκοπός μας δεν είναι μόνο να σχεδιάσουμε έναν αλγόριθμο, αλλά και να διαπιστώσουμε την επίδοσή του, δηλαδή να εκφράσουμε την πολυπλοκότητά του με το συμβολισμό O. Στη συνέχεια θα εξετάσουμε λεπτομερέστερα δύο από τους αλγορίθμους που αναπτύχθηκαν στο κεφάλαιο 3, ενώ για τους υπόλοιπους θα δώσουμε απλώς την αντίστοιχη έκφραση με βάση το συμβολισμό O.

5.3.1 Ταξινόμηση ευθείας ανταλλαγής

Το πιο βασικό κριτήριο της επίδοσης μίας μεθόδου ταξινόμησης είναι ο αριθμός C, που μετρά τις απαιτούμενες *συγκρίσεις κλειδιών* (key comparisons), που εκτελούνται μέχρι να τελειώσει η ταξινόμηση. Ένα άλλο κριτήριο είναι ο αριθμός M που μετρά τις *μετακινήσεις* (moves) των στοιχείων. Οι αριθμοί C και M είναι συναρτήσεις του αριθμού n των στοιχείων, που πρέπει να ταξινομηθούν.

Με βάση τον αλγόριθμο όπως περιγράφεται στην παράγραφο, εύκολα προκύπτει ότι ο αριθμός των συγκρίσεων στην καλύτερη, τη μέση και τη χειρότερη περίπτωση είναι ο ίδιος. Ο αριθμός αυτός είναι:

$$C = 1 + 2 + \ldots + (n-1)$$

Με βάση τις ιδιότητες της αριθμητικής προόδου προκύπτει ότι

$$C = \frac{n(n-1)}{2}$$

που τελικά σημαίνει, ότι η πολυπλοκότητα της ταξινόμησης αυτής είναι $O(n^2)$.

5.3.2 Γραμμική αναζήτηση

Στην παράγραφο 3.6 μελετήσαμε την απλούστερη μέθοδο αναζήτησης, τη γραμμική ή σειριακή μέθοδο. Όταν αναζητούμε ένα κλειδί που πράγματι υπάρχει στον πίνακα, τότε λέμε ότι η αναζήτηση είναι *επιτυχής* (successful). Στην αντίθετη περίπτωση, λέμε ότι η αναζήτηση είναι *ανεπιτυχής* (unsuccessful). Το κόστος της αναζήτησης μετράται με τον αριθμό των συγκρίσεων κλειδιών. Έτσι το κόστος αυτό για την επιτυχή αναζήτηση συμβολίζεται με Ε, ενώ για την ανεπιτυχή αναζήτηση συμβολίζεται με Α.

Ας εξετάσουμε αρχικά, πως προκύπτει η πολυπλοκότητα της επιτυχούς αναζήτησης σε μη ταξινομημένο πίνακα που αποτελείται από n στοιχεία. Αν αναζητάται το πρώτο στοιχείο, τότε η επιτυχής αναζήτηση θα κοστίσει μία σύγκριση, ενώ αν αναζητάται το δεύτερο στοιχείο, τότε το κόστος είναι δύο συγκρίσεις. Με την ίδια λογική, αν αναζητάται το n-οστό στοιχείο, τότε θα εκτελεσθούν n συγκρίσεις κλειδιών μέχρι την περάτωση του αλγορίθμου. Έτσι κατά μέσο όρο, ο απαιτούμενος αριθμός συγκρίσεων κλειδιών για την επιτυχή αναζήτηση είναι:

$$E = \frac{(1+2+\ldots+n)}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

Από τη σχέση αυτή, εύκολα καταλήγουμε στο συμπέρασμα ότι η επιτυχής αναζήτηση έχει πολυπλοκότητα της τάξης O(n), με την απαραίτητη προϋπόθεση ότι τα κλειδιά αναζητώνται ισοπίθανα.

Η πολυπλοκότητα της ανεπιτυχούς αναζήτησης είναι επίσης τάξης O(n). Αυτό προκύπτει με βάση την απλή σκέψη ότι, όταν το αναζητούμενο κλειδί δεν υπάρχει στον πίνακα, τότε η αναζήτηση καταλήγει να εξετάσει ένα προς ένα όλα τα κλειδιά μέχρι το τέλος του πίνακα. Αν ο πίνακας είναι ταξινομημένος, τότε η διαδικασία της επιτυχούς και της ανεπιτυχούς αναζήτησης μπορεί να βελτιωθεί, ωστόσο και πάλι η πολυπλοκότητα θα είναι γραμμικής τάξης.

Πιν. 5.4. Πολυπλοκότητες μερικών αλγορίθμων

Αλγόριθμος	Πολυπλοκότητα
Ώθηση-απώθηση σε στοίβα	O(1)
Εισαγωγή-εξαγωγή σε ουρά	O(1)
Fibonacci επαναληπτική	O(n)
Ταξινόμηση ευθείας ανταλλαγής	$O(n^2)$
Σειριακή αναζήτηση	O(n)
Δυαδική αναζήτηση	O(logn)

5.4 Είδη αλγορίθμων

Όπως αναφέρθηκε ήδη και όπως φάνηκε από τα προηγούμενα, το αντικείμενο των αλγορίθμων έχει μεγάλο πλάτος και βάθος, μεγάλη ιστορία και μεγάλο μέλλον, καθώς είναι η βάση όπου στηρίζονται όλες σχεδόν οι υπόλοιποι τομείς της Πληροφορικής. Έχουν αναπτυχθεί, λοιπόν, πολλά είδη και κατηγορίες αλγορίθμων. Για παράδειγμα, μεταξύ των άλλων στα προηγούμενα κεφάλαια μιλήσαμε για αναδρομικούς και επαναληπτικούς αλγορίθμους. Στη συνέχεια θα αναφερθούμε σε μερικές νέες έννοιες σχετικά με τις κατηγορίες των αλγορίθμων.

Είναι γενικά γνωστό ότι, υπάρχουν σήμερα ακριβοί υπολογιστές με δυνατότητες που ξεπερνούν τα όρια των προσωπικών/οικιακών υπολογιστών. Οι υπολογιστές αυτοί χρησιμοποιούνται από μεγάλες επιχειρήσεις, οργανισμούς και ιδρύματα, και διακρίνονται από τη συνθετότητά τους, αφού αποτελούνται από πολλούς επεξεργαστές, πολλές κύριες μνήμες και πολλές δευτερεύουσες μνήμες (μαγνητικούς δίσκους). Στους υπολογιστές αυτούς είναι δυνατόν ένας αλγόριθμος να κατατμηθεί σε μικρότερα κομμάτια που εκτελούνται παράλληλα, αφού απαιτούν διαφορετικούς υπολογιστικούς πόρους. Σε αυτά τα υπολογιστικά συστήματα ο χρόνος που απαιτείται από έναν αλγόριθμο, είναι κλάσμα του απαιτούμενου χρόνου από τον ίδιο αλγόριθμο σε έναν προσωπικό υπολογιστή, που τυπικά διαθέτει έναν επεξεργαστή, μία κύρια μνήμη και μία δευτερεύουσα μνήμη. Η ανάπτυξη *παράλληλων* (parallel) αλγορίθμων για περιβάλλοντα, όπως αυτό που προαναφέρθηκε, είναι μία σημαντική περιοχή που απασχολεί πλήθος επιστημόνων, καθώς διαφαίνεται ότι στο μέλλον οι υπολογιστές αυτοί θα γίνουν φθηνότεροι και θα επεκταθεί η χρήση τους. Ωστόσο, στα πλαίσια του μαθήματος αυτού δεν θα μας απασχολήσει η σχεδίαση και η ανάλυση παράλληλων αλγορίθμων.

Για να γίνει καλύτερα κατανοητή η έννοια του παράλληλου αλγόριθμου, ας θεωρήσουμε και πάλι το αρχικό παράδειγμα του 2ου κεφαλαίου, όπου πρέπει να δρομολογήσουμε τις ενέργειές μας, ώστε να γευματίσουμε. Όμως τώρα θα θεωρήσουμε ότι εργάζονται δύο άτομα. Έτσι, ο επόμενος πίνακας δίνει τις ενέργειες του κάθε ατόμου.

Πρώτο άτομο	Δεύτερο άτομο
Προετοιμασία σκευών μαγειρικής	Ετοιμασία σαλάτας
Παρασκευή φαγητού	Στρώσιμο τραπεζιού
Γεύμα	Γεύμα
Καθαριότητα τραπεζιού	Πλύσιμο πιάτων και κουζινικών

Παρατηρούμε ότι τώρα η διαδικασία ολοκληρώνεται σε λιγότερο χρόνο, αφού σε κάθε άτομο αναλογούν μόνο τέσσερις εργασίες.

Γενικά, η έννοια του παραλληλισμού υπάρχει παντού γύρω μας. Για παράδειγμα, στην τράπεζα πολλοί ταμίες εξυπηρετούν ταυτόχρονα τους πελάτες που δημιουργούν μία ή περισσότερες ουρές, στο θέατρο υπάρχουν πολλές έξοδοι, ώστε ο θεατές να φεύγουν γρήγορα και με ασφάλεια, στο πρατήριο βενζίνης υπάρχουν πολλές αντλίες, ώστε να μειώνεται ο χρόνος αναμονής των αυτοκινήτων κ.λπ.

Επειδή η βελτίωση των αλγορίθμων είναι ζωτικής σημασίας, μία ιδιαίτερη περιοχή της Πληροφορικής, η *Υπολογιστική Πολυπλοκότητα* (Computational Complexity), διερευνά από την αναλυτική άποψη τους αλγορίθμους και τους ταξινομεί ανάλογα με την επίδοσή τους. Έτσι συνεχώς νέοι καλύτεροι αλγόριθμοι αναπτύσσονται, ενώ άλλοι εγκαταλείπονται ως μη αποτελεσματικοί. Ένας αλγόριθμος λέγεται *άριστος* (optimal), αν αποδειχθεί ότι είναι τόσο αποτελεσματικός, ώστε δεν μπορεί να κατασκευασθεί καλύτερος.

Πολυωνυμικοί (polynomial) λέγονται οι αλγόριθμοι με πολυπλοκότητα που φράσσεται από επάνω με μία πολυωνυμική έκφραση. Για παράδειγμα, πολυωνυμικοί είναι οι αλγόριθμοι τάξης $O(n)$, $O(n^{3/2})$, $O(n^2)$ κ.λπ. Συνήθως αυτοί δεν απαιτούν μεγάλη υπολογιστική προσπάθεια σε αντίθεση με τους αλγορίθμους πολυπλοκότητας τάξης $O(2^n)$, $O(n^2 2^n)$ ή $O(n^n)$, που ονομάζονται *μη πολυωνυμικοί ή εκθετικοί*.

Πολλές φορές μερικά προβλήματα σε πρώτη εξέταση μοιάζουν παρόμοια, αλλά στη συνέχεια μπορεί να αποδειχθεί ότι έχουν εξαιρετικά διαφορετικό βαθμό δυσκολίας ως προς την εύρεση αποτελεσματικής λύσης. Για παράδειγμα, έστω τα εξής δύο προβλήματα:

- Δίδεται ένα σύνολο διαφορετικών θετικών αριθμών $x_1, x_2, \ldots, x_n$ (όπου το n άρτιος αριθμός). Να βρεθούν δύο υποσύνολα από n/2 αριθμούς, όπου η διαφορά των αθροισμάτων κάθε υποσυνόλου να είναι μέγιστη,
- Δίδεται ένα σύνολο διαφορετικών θετικών αριθμών $x_1, x_2, \ldots, x_n$ (όπου το n άρτιος αριθμός). Να βρεθούν δύο υποσύνολα από n/2 αριθμούς, όπου η διαφορά των αθροισμάτων κάθε υποσυνόλου να είναι ελάχιστη.

Σε σχέση με το πρώτο πρόβλημα, εύκολα μπορούμε να προτείνουμε αποτελεσματικούς αλγορίθμους. Για παράδειγμα, μπορούμε να ταξινομήσουμε τους n αριθμούς και να τους χωρίσουμε σε δύο ομάδες, τους μικρούς και τους μεγάλους. Η λύση αυτή έχει πολυπλοκότητα $O(n^2)$, αν χρησιμοποιηθεί η ταξινόμηση φυσσαλίδας. Μάλιστα σημειώνεται ότι, μπορεί να προταθεί και ακόμη πιο αποτελεσματικός αλγόριθμος γραμμικής πολυπλο-

κότητας. Ωστόσο, σε σχέση με το δεύτερο πρόβλημα δεν υπάρχει αποτελεσματικός αλγόριθμος. Η μοναδική λύση στηρίζεται στην εξαντλητική δοκιμή όλων των δυνατών τρόπων. Ένας τέτοιος αλγόριθμος είναι πολυπλοκότητας $O(2^n)$.

Κατά παρόμοιο τρόπο υπάρχουν πολλά προβλήματα που δεν μπορούν να επιλυθούν με κάποιο αποτελεσματικό (δηλαδή πολυωνυμικό) αλγόριθμο, αλλά πρέπει να δοκιμασθούν όλες οι δυνατές περιπτώσεις ώστε να επιλεχθεί η καλύτερη. Τα προβλήματα αυτά ονομάζονται *δυσχείριστα* (intractable). Ευρύτατα γνωστό ως δυσχείριστο πρόβλημα είναι το πρόβλημα της συντομότερης διαδρομής που πρέπει ένας περιοδεύων πωλητής να ακολουθήσει, ώστε να περάσει μία και μόνο μία φορά από κάθε πόλη και να επιστρέψει στην αρχική. Το πρόβλημα αυτό αναφέρθηκε ως παράδειγμα στην αρχή του κεφαλαίου. Ένα άλλο γνωστό δυσχείριστο πρόβλημα είναι ο χρωματισμός ενός γράφου. Σύμφωνα με το πρόβλημα αυτό, πρέπει να χρωματίσουμε τις κορυφές ενός γράφου χρησιμοποιώνας ένα συγκεκριμένο αριθμό χρωμάτων, έτσι ώστε δύο γειτονικές κορυφές ή περιοχές να μην έχουν το ίδιο χρώμα. Δυσχείριστο επίσης είναι το πρόβλημα του χρωματισμού των συνδέσεων μεταξύ των ακμών ενός γράφου, έτσι ώστε οι συνδέσεις που καταλήγουν σε μία συγκεκριμένη κορυφή να είναι διαφορετικού χρώματος.

Τα δυσχείριστα αυτά προβλήματα, που αποκαλούνται και *ανοικτά* προβλήματα, λέγεται ότι ανήκουν στην κατηγορία των προβλημάτων *NP-πλήρων* προβλημάτων από τα αρχικά των αγγλικών όρων *Nondeterministic Polynomial*. Για κάθε πρόβλημα της κατηγορίας αυτής δεν υπάρχει ή τουλάχιστον δεν έχει βρεθεί ακόμη αλγόριθμος επίλυσής τους σε πολυωνυμικό χρόνο.

Η αδυναμία της επιστήμης να προτείνει αποτελεσματικούς αλγορίθμους για πολλά δυσχείριστα προβλήματα, έχει αναγκαστικά οδηγήσει στην ανάπτυξη *προσεγγιστκών* (approximate) αλγορίθμων για την επίλυσή τους, έτσι ώστε να επιτυγχάνεται μία αποδεκτή λύση σε λογικό χρόνο. Λέγοντας *αποδεκτή* εννοούμε ότι η λύση αυτή, ναι μεν δεν είναι η καλύτερη δυνατή, ωστόσο δίνει στο πρόβλημα μία απάντηση που πλησιάζει την καλύτερη λύση σε μεγάλο βαθμό.

Ευριστικός είναι ο αλγόριθμος που είτε μπορεί να οδηγήσει σε μία καλή ή ακόμη και βέλτιστη λύση ενός προβλήματος, αν είμαστε τυχεροί, αλλά μπορεί να οδηγήσει και σε μία λύση που απέχει πολύ από τη βέλτιστη ή ακόμη, και να αποτύχει να βρει λύση, αν είμαστε άτυχοι. Οι ευριστικοί αλγόριθμοι δεν είναι τυποποιημένοι και στηρίζονται σε μια εμπειρική παρατήρηση ή ένα τέχνασμα ή μία έμπνευση του προγραμματιστή. Συνήθως οι προσεγγι-

στικοί αλγόριθμοι είναι ευριστικοί αλγόριθμοι, αλλά βέβαια υπάρχουν και πολλοί ευριστικοί αλγόριθμοι που δεν είναι προσεγγιστικοί.

Ανακεφαλαίωση

Αρχικά ορίστηκε και αναλύθηκε η έννοια της επίδοσης των αλγορίθμων, με ειδικότερη αναφορά στη χειρότερη περίπτωση κάθε αλγόριθμου.

Έγινε καταγραφή των μεγεθών που επηρεάζουν την επίδοση ενός αλγορίθμου και των βασικών πράξεων για την ταξινόμηση, την αναζήτηση και τον πολλαπλασιασμό καθώς και για άλλα είδη προβλημάτων.

Δόθηκε ορισμός για την αποδοτικότητα και για την ορθότητα των αλγορίθμων. Η πολυπλοκότητα αλγορίθμων περιγράφεται αναλυτικά και τυποποιούνται οι ορισμοί για τις κατηγορίες (τάξεις) πολυπλοκότητας των περισσότερων ειδών αλγορίθμων.

Μερικοί από τους πλέον γνωστούς αλγορίθμους αναλύονται και εκφράζεται η πολυπλοκότητα τους με χρήση των αντίστοιχων συμβολισμών των κατηγοριών πολυπλοκότητας.

Τέλος, παρουσιάζεται καταγραφή των ειδών των αλγορίθμων και τυποποιούνται με χρήση ορισμών οι πλέον γνωστοί τύποι αλγορίθμων (πολυωνυμικοί, μη πολυωνυμικοί, παράλληλοι αλγόριθμοι).

Λέξεις κλειδιά

Επίδοση/αποδοτικότητα αλγορίθμου, Χειρότερη περίπτωση αλγορίθμου, Μέγεθος εισόδου, Ορθότητα αλγορίθμου, Πολυπλοκότητα αλγορίθμων, Συμβολισμός Ο, Ανάλυση αλγορίθμων, Είδη αλγορίθμων, Υπολογιστική Πολυπλοκότητα, Προσεγγιστικοί αλγόριθμοι, Ευριστικοί αλγόριθμοι

Ερωτήσεις - Θέματα για συζήτηση

1. Ποιά είναι τα πρωταρχικά ερωτήματα που τίθενται για την κατανόηση της επίδοσης ενός αλγορίθμου;
2. Τι είναι και πώς μπορεί να εκφρασθεί η χειρότερη περίπτωση ενός αλγορίθμου;
3. Να περιγραφεί το μέγεθος της εισόδου ενός αλγορίθμου και να δοθεί ένα παράδειγμα αναγνώρισης και καταγραφής αυτού του μεγέθους.

4. Από ποιους κύριους παράγοντες εξαρτάται ο χρόνος εκτέλεσης ενός αλγορίθμου;
5. Ποιες είναι οι απαραίτητες προϋποθέσεις για να είναι δυνατή η σύγκριση μεταξύ δύο προγραμμάτων αλγορίθμων;
6. Να ορισθεί η αποδοτικότητα αλγορίθμων.
7. Τι είναι ο έλεγχος ορθότητας ενός αλγορίθμου και ποιοι περιορισμοί υπάρχουν για αυτόν τον έλεγχο;
8. Να περιγραφεί ο τρόπος απόδειξης της ορθότητας ενός αλγορίθμου και να δοθεί ένα παράδειγμα απόδειξης ορθότητας αλγορίθμου.
9. Να δοθεί ο ορισμός της πολυπλοκότητας αλγορίθμου.
10. Με τη βοήθεια ενός διαγράμματος να γίνει σύγκριση μεταξύ της λογαριθμικής, της γραμμικής και της τετραγωνικής πολυπλοκότητας.
11. Ποια είναι η πολυπλοκότητα των λειτουργιών σε στοίβες και ουρές;
12. Ποια είναι η πολυπλοκότητα της σειριακής αναζήτησης;
13. Ποια είναι τα κυριότερα είδη αλγορίθμων;

Βιβλιογραφία

1. Φώτω Αφράτη και Γιώργος Παπαγεωργίου, *Αλγόριθμοι – Μέθοδοι Σχεδίασης και Ανάλυση Πολυπλοκότητας*, Εκδόσεις Συμμετρία, Αθήνα, 1993.
2. Χρήστος Κοίλιας, *Δομές Δεδομένων και Οργάνωση Αρχείων*. Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1993.
3. Μανόλης Λουκάκης, *Δομές Δεδομένων και Αλγόριθμοι*, Εκδόσεις Θεραπευτικής Κοινότητας Ιθάκη, Θεσσαλονίκη, 1988.
4. Ιωάννης Μανωλόπουλος, *Δομές Δεδομένων – μία Προσέγγιση με Pascal*, Εκδόσεις Art of Text, Θεσσαλονίκη, 1998.
5. Ν. Μισυρλής, *Δομές Δεδομένων*, Συμμετρία, Αθήνα, 1993.
6. Niklaus Wirth, *Αλγόριθμοι και Δομές Δεδομένων*, Κλειδάριθμος, Αθήνα, 1990.
7. G. Brassard and P.Bratley, *Fundamentals of Algorithms*, Prentice Hall, 1996.
8. T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms* MIT Press, 1990.

9. E. Horowitz, S. Sahni and S. Rajasekaran, *Computer Algorithms*, Computer Science Press, 1998.

10. I. Oliver, *Programming Classics: Implementing the World's Best Algorithms*, Prentice Hall, 1993.

Διευθύνσεις Διαδικτύου

- http://www.ioinformatics.org/

Κόμβος Ολυμπιάδων Πληροφορικής με σχετικά προβλήματα αλγορίθμων.

ΚΕΦΑΛΑΙΟ 6ο

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

ΠΕΡΙΕΧΟΜΕΝΑ

- Η έννοια του προγράμματος
- Ιστορική αναδρομή
- Φυσικές και τεχνητές γλώσσες
- Τεχνικές σχεδίασης προγραμμάτων
- Αντικειμενοστραφής προγραμματισμός
- Παράλληλος προγραμματισμός
- Προγραμματιστικά περιβάλλοντα

Εισαγωγή

Στα προηγούμενα κεφάλαια αναφερθήκαμε αναλυτικά στην ανάπτυξη των αλγορίθμων και των διαφόρων τεχνικών. Στα επόμενα κεφάλαια θα ασχοληθούμε με τον προγραμματισμό δηλαδή τη διατύπωση των αλγορίθμων σε τέτοια μορφή, ώστε να μπορούν να υλοποιηθούν από τον υπολογιστή. Το κεφάλαιο αυτό ασχολείται με γενικές έννοιες που είναι απαραίτητες πριν από την ενασχόληση με τη διαδικασία του προγραμματισμού. Ορίζεται η έννοια του προγράμματος, παρουσιάζεται μία σύντομη ιστορία των γλωσσών προγραμματισμού και των ειδών προγραμματισμού, καθώς και οι τεχνικές που χρησιμοποιούνται για τη σωστή δημιουργία προγραμμάτων. Συγκεκριμένα παρουσιάζονται οι τεχνικές της ιεραρχικής σχεδίασης προγραμμάτων, του τμηματικού προγραμματισμού και κύρια οι αρχές του δομημένου προγραμματισμού. Επίσης περιγράφεται η διαδικασία που ακολουθείται από τη τη σύνταξη του προγράμματος μέχρι την τελική του εκτέλεση από τον υπολογιστή και τη λήψη των αποτελεσμάτων.

Διδακτικοί στόχοι

Στόχοι του κεφαλαίου αυτού είναι ο μαθητής :

- Να ορίζει τι είναι πρόγραμμα και να κατατάσσει και να συγκρίνει τις γλώσσες προγραμματισμού.
- Να αναγνωρίζει τα κυριότερα είδη προγραμματισμού και να περιγράφει τα βασικά χαρακτηριστικά των τεχνικών που χρησιμοποιούνται.
- Να διατυπώνει τα πλεονεκτήματα του δομημένου προγραμματισμού.
- Να περιγράφει τη διαδικασία εκτέλεσης ενός προγράμματος.
- Να αναφέρει τα βασικά προγράμματα που περιέχει ένα προγραμματιστικό περιβάλλον

Προερωτήσεις

- ✓ Η δημιουργία του αλγορίθμου αρκεί για να επιλύσουμε ένα πρόβλημα στον υπολογιστή;
- ✓ Πώς διαχειρίζεται τις πληροφορίες ο υπολογιστής;
- ✓ Γνωρίζεις κάποιες γλώσσες προγραμματισμού;
- ✓ Πώς και γατί εξελίσσονται οι γλώσσες;
- ✓ Η ύπαρξη συγκεκριμένων μεθοδολογιών και τεχνικών βοηθάει στην επίλυση των προβλημάτων;

6.1 Η έννοια του προγράμματος

Η επίλυση ενός προβλήματος με τον υπολογιστή περιλαμβάνει, όπως έχει ήδη αναφερθεί, τρία εξίσου σημαντικά στάδια.

- Τον ακριβή προσδιορισμό του προβλήματος.
- Την ανάπτυξη του αντίστοιχου αλγορίθμου.
- Τη διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή.

Ο προγραμματισμός ασχολείται με το τρίτο αυτό στάδιο, τη δημιουργία του προγράμματος δηλαδή του συνόλου των εντολών που πρέπει να δοθούν στον υπολογιστή, ώστε να υλοποιηθεί ο αλγόριθμος για την επίλυση του προβλήματος. Το πρόγραμμα, το οποίο γράφεται σε κάποια γλώσσα προγραμματισμού, δεν είναι απλά η υλοποίηση του αλγορίθμου, αλλά βασικό στοιχείο του είναι τα δεδομένα και οι δομές δεδομένων επί των οποίων ενεργεί. Αναφέρθηκε ήδη ότι οι αλγόριθμοι και οι δομές δεδομένων είναι μια αδιάσπαστη ενότητα.

Ο προγραμματισμός είναι αυτός που δίνει την εντύπωση ότι, οι υπολογιστές είναι έξυπνες μηχανές που επιλύουν τα πολύπλοκα προβλήματα.

Οι γλώσσες προγραμματισμού αναπτύχθηκαν με σκοπό την επικοινωνία του ανθρώπου (προγραμματιστή) με τη μηχανή (υπολογιστή)

Η εντύπωση αυτή όμως είναι απλώς μία ψευδαίσθηση. Ο υπολογιστής, ως γνωστόν, είναι μία μηχανή που καταλαβαίνει μόνο δύο καταστάσεις, οι οποίες αντιπροσωπεύονται με δύο αριθμούς το μηδέν και το ένα, τα ψηφία του δυαδικού συστήματος. Το μόνο πράγμα που κάνει ο υπολογιστής είναι στοιχειώδεις ενέργειες σε ακολουθίες αυτών των δύο ψηφίων, αλλά αυτές τις ενέργειες τις εκτελεί μα ασύλληπτη ταχύτητα. Ο υπολογιστής μπορεί απλά να αποθηκεύει στη μνήμη τις ακολουθίες των δυαδικών ψηφίων, να τις ανακτά, να κάνει στοιχειώδεις αριθμητικές πράξεις με αυτές και να τις συγκρίνει.

6.2 Ιστορική αναδρομή

Από τη δημιουργία του πρώτου υπολογιστή μέχρι σήμερα έχουν αλλάξει πάρα πολλά πράγματα. Οι πρώτοι υπολογιστές, τεράστιοι σε μέγεθος αλλά με πάρα πολύ περιορισμένες δυνατότητες και μικρές ταχύτητας επεξεργασίας εξελίχθηκαν σε πολύ μικρούς σε μέγεθος υπολογιστές με τεράστιες όμως δυνατότητες και ταχύτητες επεξεργασίας.

Ενώ λοιπόν το υλικό (hardware) των υπολογιστών βελτιώνεται, τελειοποιείται και ταυτόχρονα παρέχει νέες δυνατότητες επεξεργασίας, οι βασι-

κές αρχές λειτουργίας των υπολογιστών που διατυπώθηκαν το μακρινό 1945 από τον Φον Νόυμαν, δεν άλλαξαν πρακτικά καθόλου. Την ίδια αργή εξέλιξη ουσιαστικά έχουν και οι γλώσσες προγραμματισμού, οι οποίες αν και εξελίσσονται και συνεχώς εμπλουτίζονται με νέες δυνατότητες, τα χαρακτηριστικά τους και οι βασικές τους ιδιότητες ουσιαστικά παραμένουν τα ίδια.

6.2.1 Γλώσσες μηχανής

*Ενα πρόγραμμα σε **γλώσσα μηχανής** είναι μια ακολουθία δυαδικών ψηφίων, που αποτελούν εντολές προς τον επεξεργαστή για στοιχειώδεις λειτουργίες.*

Αρχικά για να μπορέσει ο υπολογιστής να εκτελέσει μία οποιαδήποτε λειτουργία, έπρεπε να δοθούν κατευθείαν οι κατάλληλες ακολουθίες από 0 και 1, δηλαδή εντολές σε μορφή κατανοητή από τον υπολογιστή αλλά ακατανόητες από τον άνθρωπο. Ο τρόπος αυτός ήταν επίπονος και ελάχιστοι μπορούσαν να τον υλοποιήσουν, αφού απαιτούσε βαθιά γνώση του υλικού και της αρχιτεκτονικής του υπολογιστή.

Ο πρώτος υπολογιστής ο περίφημος ENIAC για να "προγραμματιστεί", ώστε να εκτελέσει κάποιους υπολογισμούς, έπρεπε να αλλάξουν θέση εκατοντάδες διακόπτες και να ρυθμιστούν αντίστοιχα όλες οι καλωδιώσεις, διαδικασία εξαιρετικά επίπονη και χρονοβόρα. Ο "προγραμματισμός" των πρώτων αυτών υπολογιστών, δεν ήταν ουσιαστικά προγραμματισμός με τη σημερινή έννοια του όρου. Ο υπολογιστής αναδιαρθρωνόταν, ώστε να εκτελέσει τους απαιτούμενους υπολογισμούς και στη συνέχεια έπρεπε να αλλάξει πάλι η διάρθρωσή του, ώστε να εκτελέσει έναν άλλο υπολογισμό.

Οι εντολές ενός προγράμματος και σήμερα μετατρέπονται σε ακολουθίες που αποτελούνται από 0 και 1, τις εντολές σε **γλώσσα μηχανής,** όπως ονομάζονται, οι οποίες εκτελούνται από τον υπολογιστή.

6.2.2 Συμβολικές γλώσσες ή γλώσσες χαμηλού επιπέδου

Από τα πρώτα χρόνια άρχισαν να γίνονται προσπάθειες για τη δημιουργία μίας συμβολικής γλώσσας, η οποία ενώ θα έχει έννοια για τον άνθρωπο, θα μετατρέπεται εσωτερικά από τους υπολογιστές στις αντίστοιχες ακολουθίες από 0 και 1. Για παράδειγμα η λέξη ADD (πρόσθεσε) ακολουθούμενη από δύο αριθμούς, είναι κατανοητή από τον άνθρωπο και απομνημονεύεται σχετικά εύκολα. Η εντολή αυτή θα μεταφραστεί από τον υπολογιστή σε μία ακολουθία δυαδικών ψηφίων και στη συνέχεια μπορεί να εκτελεστεί. Το έργο της μετάφρασης το αναλαμβάνει ένα ειδικό πρόγραμμα, ο **συμβολομεταφραστής** (assembler).

Η χρήση των πρώτων αυτών συμβολικών γλωσσών, που συνεχίζουν να χρησιμοποιούνται για ειδικούς σκοπούς, ήταν σαφώς μια εξέλιξη από τις α-

κατανόητες ακολουθίες δυαδικών στοιχείων. Ωστόσο παρέμεναν στενά συνδεδεμένες με την αρχιτεκτονική του κάθε υπολογιστή. Επίσης δεν διέθεταν εντολές πιο σύνθετων λειτουργιών οδηγώντας έτσι σε μακροσκελή προγράμματα, που ήταν δύσκολο να γραφούν και κύρια να συντηρηθούν. Ακόμη τα προγράμματα δεν μπορούν να μεταφερθούν σε άλλον διαφορετικό υπολογιστή, ακόμη και του ίδιου κατασκευαστή. Οι γλώσσες αυτές ονομάζονται συμβολικές ή γλώσσες χαμηλού επιπέδου, αφού εξαρτώνται από την αρχιτεκτονική του υπολογιστή.

Οι εντολές σε συμβολική γλώσσα αποτελούνται από συμβολικά ονόματα που αντιστοιχούν σε εντολές της γλώσσας μηχανής. Οι συμβολικές γλώσσες είναι συνδεδεμένες με την αρχιτεκτονική κάθε υπολογιστή

```
10101000 00001010          INDEX=$01     sum = 0
10001100 00000001          SUM=$02       FOR index=1 TO 10
00111100                   LDA #10           sum=sum+index
01010001 00000001          STA INDEX     NEXT index
01000011 00000001          CLA           END
11000000 11111010    LOOP  ADD INDEX
10001100 00000010          DEC INDEX
11111111                   BNE LOOP
                           STA SUM
                           BRK
```

Σχ. 6.1. *Ενα πρόγραμμα σε γλώσσα μηχανής, σε συμβολική γλώσσα και σε γλώσσα υψηλού επιπέδου για τον υπολογισμό του αθροίσματος των αριθμών 1 έως 10*

6.2.3 Γλώσσες υψηλού επιπέδου

Οι παραπάνω ανεπάρκειες των συμβολικών γλωσσών και η προσπάθεια για καλύτερη επικοινωνία ανθρώπου-μηχανής, οδήγησαν στα τέλη της δεκαετίας του 50 στην εμφάνιση των πρώτων γλωσσών προγραμματισμού υψηλού επιπέδου.

Το 1957 η IBM ανάπτυξε την πρώτη γλώσσα υψηλού επιπέδου τη **FORTRAN.** Το όνομα FORTRAN προέρχεται από τις λέξεις FORmula TRANslation, που σημαίνουν μετάφραση τύπων. Η FORTRAN αναπτύχθηκε ως γλώσσα κατάλληλη για την επίλυση μαθηματικών και επιστημονικών προβλημάτων. Το πρόγραμμα που γράφεται σε FORTRAN ή σε οποιαδήποτε άλλη γλώσσα υψηλού επιπέδου, μεταφράζεται από τον ίδιο τον υπολογιστή στις ακολουθίες των εντολών της μηχανής με τη βοήθεια ενός ειδικού προγράμματος, που ονομάζεται μεταγλωττιστής. Το ίδιο πρόγραμμα FORTRAN μπορεί να εκτελεστεί σε οποιοδήποτε άλλο υπολογιστή, αρκεί να υπάρχει ο αντίστοιχος μεταγλωττιστής για τον υπολογιστή αυτό. Η γλώσσα FORTRAN μετά από πολλές αλλαγές, προσθήκες και βελτιώσεις χρησιμοποιείται ακόμη και σήμερα για επιστημονικές εφαρμογές.

```
C   PROGRAM EQUATION
    READ(*,1) A,B
1   FORMAT(F5.1)
    IF (A.EQ.0) GO TO 20
    X=(-1.)*B/A
    WRITE(*,2) X
2   FORMAT('X=',F10.2)
    GO TO 50
20  IF (B.EQ.0) WRITE(*,3)
    IF (B.NE.0) WRITE(*,4)
3   FORMAT('ΑΟΡΙΣΤΗ')
4   FORMAT('ΑΔΥΝΑΤΗ')
50  STOP
    END
```

Σχ.6.2. *Η γλώσσα FORTRAN υπήρξε η πρώτη γλώσσα προγραμματισμού υψηλού επιπέδου. Πρόκειται για γλώσσα κατάλληλη για υπολογισμούς, ενώ υστερεί στη διαχείριση αρχείων δεδομένων και γενικότερα αλφαριθμητικών πληροφοριών. Γνώρισε πολλές βελτιώσεις με κυριότερους σταθμούς τις εκδόσεις 4, 77, 90/95 και Visual FORTRAN. Το πρόγραμμα του παραδείγματος επιλύει την εξίσωση α' βαθμού.*

Η FORTRAN παρά τα ισχυρά χαρακτηριστικά της και τις συνεχείς αλλαγές που τη καθιστούσαν συνεχώς αποτελεσματικότερη, δεν μπορούσε να καλύψει τις απαιτήσεις σε όλους τους τομείς δραστηριοτήτων, όπως και καμία άλλη γλώσσα προγραμματισμού δεν κατάφερε. Έτσι αναπτύχθηκαν και συνεχίζουν να αναπτύσσονται πολλές γλώσσες προγραμματισμού για διάφορες περιοχές δραστηριοτήτων.

Το 1960 αναπτύχθηκε μία άλλη γλώσσα, σταθμός στον προγραμματισμό η γλώσσα COBOL. Η **COBOL** όπως δηλώνει και το όνομα της (COmmon Business Oriented Language - Κοινή γλώσσα προσανατολισμένη στις επιχειρήσεις) είναι κατάλληλη για ανάπτυξη εμπορικών εφαρμογών, και γενικότερα διαχειριστικών εφαρμογών, τομέας όπου η FORTRAN υστερούσε. Η COBOL καθιερώθηκε ως πρότυπο και χρησιμοποιήθηκε από πολλές επιχειρήσεις καθώς και από όλη τη δημόσια διοίκηση. Η γλώσσα γνώρισε πολλές εκδόσεις και πάρα πολλές εφαρμογές βρίσκονται σε χρήση ακόμη και σήμερα.

Μια από τις σημαντικότερες γλώσσα προγραμματισμού με ελάχιστη πρακτική εφαρμογή αλλά που επηρέασε ιδιαίτερα τον προγραμματισμό και τις επόμενες γλώσσες, είναι η **ALGOL** (ALGOrithmic Language – Αλγοριθμική γλώσσα). Αναπτύχθηκε από Ευρωπαίους επιστήμονες, αρχικά το

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  EQUATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC.
SPECIAL-NAMES. DECIMAL-POINT  IS COMMA.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  X          PIC S9(6)V9.
77  A          PIC S9(6).
77  B          PIC S9(6).
77  W-X        PIC -(6),-.
PROCEDURE DIVISION.
ARXH.
    DISPLAY ' ΔΩΣΕ A'.
    ACCEPT  A.
    DISPLAY ' ΔΩΣΕ B'.
    ACCEPT  B.
    DISPLAY ' '.
    IF  A = 0   GO  TO  ROYT-1.
    COMPUTE X = B * (- 1) / A.
    MOVE  X  TO W-X.
    DISPLAY ' H ΛΥΣΗ ΕΙΝΑΙ : '  W-X.
    STOP  RUN.
ROYT-1.
    IF  B = 0
        DISPLAY ' ΑΟΡΙΣΤΗ'
    ELSE
        DISPLAY ' ΑΔΥΝΑΤΗ'.
    STOP  RUN.
```

Σχ. 6.3. *Η γλώσσα COBOL δημιουργήθηκε από την Grace Marray Hopper αξιωματικό του πολεμικού ναυτικού των ΗΠΑ το 1960. Η COBOL έκανε δυνατή την αξιοποίηση των υπολογιστών από τις επιχειρήσεις και τους οργανισμούς παρέχοντας ισχυρότατες δυνατότητες διαχείρισης αρχείων δεδομένων. Ενα πρόγραμμα COBOL διαθέτει τέσσερις υποδιαιρέσεις (divisions). Η γλώσσα χρησιμοποιεί περιγραφικό τρόπο για τη σύνταξη των εντολών με χρήση ρημάτων της αγγλικής γλώσσας, όπως ADD, MULTIPLY, MOVE κ.λπ. με αποτέλεσμα τη δημιουργία γενικά μακροσκελών προγραμμάτων. Κάθε εντολή της γλώσσας τερματίζεται με τελεία. Το πρόγραμμα του παραδείγματος επιλύει την εξίσωση α' βαθμού.*

```
10 REM ΕΠΙΛΥΣΗ ΕΞΙΣΩΣΗΣ Α'ΒΑΘΜΟΥ
20 INPUT "A=",A
30 INPUT "B=",B
40 IF A=0 THEN 100
50 X=-B/A
60 PRINT "X=";X
70 END
100 IF B=0 THEN PRINT "ΑΟΡΙΣΤΗ" ELSE PRINT "ΑΔΥΝΑΤΗ"
110 END
```

Σχ. 6.4. *Η γλώσσα BASIC δημιουργήθηκε το 1964 στο Dartmouth College από τους καθηγητές Kemeny και Kurtz. Στόχος των δημιουργών της ήταν η υλοποίηση μιας απλής γλώσσας προγραμματισμού για εκπαιδευτικούς σκοπούς. Η BASIC γνώρισε εκπληκτική διάδοση με την εμφάνιση των μικροϋπολογιστών (1975). Ο πρώτος διερμηνευτής της γλώσσας σε 8K ROM ήταν έργο των Bill Gates και Paul Allen. Ας σημειωθεί ότι, η εταιρία Microsoft ιδρύθηκε από τον Bill Gates για την εμπορική εκμετάλλευση αυτού του διερμηνευτή. Η έκδοση GWBASIC παρουσιάζεται το 1981 με τον IBM PC. Κύριο χαρακτηριστικό της γλώσσας είναι η ύπαρξη αριθμών των γραμμών του προγράμματος, οι οποίοι χρησιμοποιούνται και ως θέσεις προορισμού εντολών διακλάδωσης. Εξέλιξη της γλώσσας είναι οι εκδόσεις QuickBASIC και Visual BASIC. Το πρόγραμμα του παραδείγματος επιλύει την εξίσωση α' βαθμού.*

1960, με σκοπό τη δημιουργία γενικής φύσης προγραμμάτων που να μη συνδέονται με συγκεκριμένες εφαρμογές.

Στα μέσα της δεκαετίας του 60 αναπτύχθηκε η γλώσσα **PL/1** (Programming Language/1 – Γλώσσα Προγραμματισμού υπ΄ αριθμόν 1) που προσπάθησε, χωρίς επιτυχία να καλύψει όλους τους τομείς του προγραμματισμού, επιστημονικούς και εμπορικούς, αντικαθιστώντας τόσο τη FORTRAN όσο και την COBOL .

Στο χώρο της Τεχνητής Νοημοσύνης αναπτύχθηκαν δύο γλώσσες αρκετά διαφορετικές από όλες τις άλλες. Στα μέσα του 60 αναπτύχθηκε στο MIT η **LISP** (LISt Processor- Επεξεργαστής Λίστας), γλώσσα η οποία προσανατολίζεται σε χειρισμό λιστών από σύμβολα και η **PROLOG** (PROgramming LOGic –Λογικός Προγραμματισμός) στις αρχές του 70. Οι δύο αυτές γλώσσες χρησιμοποιούνται σε προβλήματα Τεχνητής νοημοσύνης (έμπειρα συστήματα, παιγνίδια, επεξεργασία φυσικών γλωσσών κ.λπ.).

Δύο σημαντικότατες γλώσσες γενικού σκοπού, οι οποίες αναπτύχθηκαν τη δεκαετία του 60 αλλά χρησιμοποιούνται πάρα πολύ στις ημέρες μας, είναι η BASIC και η PASCAL.

```
TO KYBOS :A
REPEAT 4 [FD :A RT 90]
PU SETPOS [20 20] PD
REPEAT 4 [FD :A RT 90]
PU HOME PD
REPEAT 2 [FD :A RT 45 FD 29 RT 135]
PU SETX :A SETY 0 PD
REPEAT 2 [FD :A RT 45 FD 29 RT 135]
HOME
END
```

Σχ. 6.5. *Η γλώσσα προγραμματισμού LOGO ολοκληρώθηκε το 1967 στη Βοστώνη από τον Seymoyr Papert. Το όνομά της προέρχεται από την ελληνική λέξη "λόγος". Πρόκειται για γλώσσα κατάλληλη για την εισαγωγή στον προγραμματισμό μαθητών μικρής ηλικίας. Το πρόγραμμα του παραδείγματος σχεδιάζει έναν κύβο ακμής Α με διαδοχικές κινήσεις της χελώνας.*

Η γλώσσα προγραμματισμού **BASIC** (Beginner's All Purpose Symbolic Instruction Code – Συμβολικός Κώδικας Εντολών Γενικής Χρήσης για Αρχάριους) αρχικά αναπτύχθηκε, όπως δηλώνει και το όνομα της, ως γλώσσα για την εκπαίδευση αρχαρίων στον προγραμματισμό. Σχεδιάστηκε για να γράφονται σύντομα προγράμματα, τα οποία εκτελούνται με τη βοήθεια διερμηνευτή (interpreter). Η ανάπτυξη όμως των μικροϋπολογιστών και οι συνεχείς εκδόσεις της γλώσσας βοήθησαν στην εξάπλωσή της, τόσο ώστε να γίνει ίσως η δημοφιλέστερη γλώσσα στους προσωπικούς υπολογιστές. Η τυποποίηση της δε από τη Microsoft με τις εκδόσεις QuickBasic και κύρια με τη Visual Basic, καθιέρωσε τη γλώσσα ως πρότυπο για ανάπτυξη εφαρμογών σε προσωπικούς υπολογιστές

Η γλώσσα **PASCAL** (δημιούργημα του καθηγητή Niklaus Wirth) έφερε μεγάλες αλλαγές στον προγραμματισμό. Παρουσιάστηκε το 1970 και στηρίχτηκε πάνω στην ALGOL. Είναι μία γλώσσα γενικής χρήσης, η οποία είναι κατάλληλη τόσο για την εκπαίδευση όσο και τη δημιουργία ισχυρών προγραμμάτων κάθε τύπου. Χαρακτηριστικό της γλώσσας είναι η η καταλληλότητα για τη δημιουργία δομημένων προγραμμάτων. Η PASCAL γνώρισε και συνεχίζει να γνωρίζει τεράστια εξάπλωση ειδικά στο χώρο των μικροϋπολογιστών και αποτέλεσε τη βάση για την ανάπτυξη άλλων ισχυρότερων γλωσσών όπως η ADA και η Modula-2.

Στα μέσα του 1960 παρουσιάστηκε για πρώτη φορά μία τεχνική σχεδίασης προγραμμάτων που έμελλε να αλλάξει ριζικά τον τρόπο ανάπτυξης προγραμμάτων καθώς και τις ίδιες τις γλώσσες προγραμματισμού. Η τεχνική του **δομημένου προγραμματισμού** η οποία εξασφαλίζει τη δημιουργία

```
(DEFUN a-exisosi (a b)
  (setf apot (- (/ b a)))
  (princ "Η εξίσωση  ")
  (princ a)
  (princ "x + ")
  (princ b)
  (princ " = 0  έχει σαν λύση χ = ")
  (princ apot))
```

Σχ. 6.6. *Η γλώσσα LISP δημιουργήθηκε το 1959 στο MIT. Πρόκειται για μη διαδικασιακή γλώσσα που προορίζεται για την επεξεργασία συμβολικών δεδομένων. Βασικός τύπος δεδομένων, από τον οποίο εξ άλλου πήρε και το όνομά της, είναι η συνδεδεμένη λίστα. Στο παράδειγμα φαίνεται μια συνάρτηση της γλώσσας, που επιλύει την εξίσωση α΄ βαθμού. Το πρόγραμμα εκτελείται δίνοντας στη γραμμή εντολών π.χ. (a-exisosi 2 5).*

προγραμμάτων απλών στη συγγραφή και την κατανόηση και εύκολων στη διόρθωση. Ο δομημένος προγραμματισμός και τα χαρακτηριστικά του θα παρουσιαστούν εκτενώς σε επόμενη παράγραφο.

Μία ακόμη γλώσσα που γνώρισε μεγάλη διάδοση είναι η γλώσσα **C**. Η C αναπτύχθηκε στα εργαστήρια της εταιρείας BELL και χρησιμοποιήθηκε για την ανάπτυξη του λειτουργικού συστήματος Unix, γλώσσα με ισχυρά χαρακτηριστικά, μερικά από αυτά κοινά με την Pascal κατάλληλη για ανάπτυξη δομημένων εφαρμογών αλλά και με πολλές δυνατότητες γλώσσας χαμηλού επιπέδου. Η C εξελίχτηκε στη γλώσσα C++, που είναι αντικειμενοστραφής. Η ιδέα του **αντικειμενοστραφούς προγραμματισμού** παρουσιάστηκε για πρώτη φορά στη δεκαετία του 70 και συνεχίζει ακόμη να απλώνεται αλλάζοντας τον παραδοσιακό προγραμματισμό. Λόγω της σημασίας του αντικειμενοστραφούς προγραμματισμού μερικά στοιχεία του παρουσιάζονται σε ξεχωριστή παράγραφο.

Τα τελευταία χρόνια χρησιμοποιείται ιδιαίτερα, ειδικά για προγραμματισμό στο Διαδίκτυο (Internet), η **JAVA.** Η JAVA είναι μία αντικειμενοστραφής γλώσσα που αναπτύχθηκε από την εταιρεία SUN με σκοπό την ανάπτυξη εφαρμογών, που θα εκτελούνται σε κατανεμημένα περιβάλλοντα, δηλαδή σε διαφορετικούς υπολογιστές οι οποίοι είναι συνδεδεμένοι στο Διαδίκτυο. Τα προγράμματα αυτά μπορούν να εκτελούνται από διαφορετικούς υπολογιστές, προσωπικούς ή μεγάλα συστήματα με διαφορετικά λειτουργικά συστήματα χωρίς αλλαγές.

Η εμφάνιση των γραφικών περιβαλλόντων εργασίας δημιούργησε την ανάγκη για ανάπτυξη προγραμμάτων που να εκμεταλλεύονται τον γραφι-

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    float a, b;

    printf("A = ");
    scanf("%f", &a);
    printf("B = ");
    scanf("%f", &b);
    if (a == 0) {
        if (b == 0) {
            printf("ΑΟΡΙΣΤΗ\n");
        }
        else {
            printf("ΑΔΥΝΑΤΗ\n");
        }
    }
    else {
        printf("X = %f\n", -b/a);
    }

    return 0;
}
```

Σχ. 6.7. *Η γλώσσα προγραμματισμού C αναπτύχθηκε στα εργαστήρια Bell της αμερικανικής εταιρίας At&T το 1972 από τον Dennis Ritchie. Η γλώσσα C είναι μια δομημένη διαδικασιακή γλώσσα γενικής χρήσης που χαρακτηρίζεται από λιτότητα στην έκφραση και μια πλούσια συλλογή τελεστών και τύπων δεδομένων. Στενά συνδεδεμένη με το UNIX, η γλώσσα C χρησιμοποιείται ευρύτατα για τη δημιουργία λειτουργικών συστημάτων και άλλων πακέτων λογισμικού. Το πρόγραμμα του παραδείγματος επιλύει την εξίσωση α' βαθμού.*

κό αυτό τρόπο επικοινωνίας χρήστη-υπολογιστή. Στα περισσότερα προγραμματιστικά περιβάλλοντα που υπήρχαν, ήταν πολύ δύσκολη έως αδύνατη η ανάπτυξη εφαρμογών, ικανών να εκμεταλλεύονται τα γραφικά αυτά χαρακτηριστικά.

Έτσι εμφανίστηκαν γλώσσες ή νέες εκδόσεις των γλωσσών που υλοποιούσαν τις έννοιες του **οδηγούμενου από το γεγονός προγραμματισμού** (event driven programming) και του **οπτικού προγραμματισμού** (visual programming).

```
CLEAR
? "1.  ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΩΝ"
? "2. ΕΚΤΥΠΩΣΗ ΕΤΙΚΕΤΩΝ"
? "3. ΠΡΟΒΟΛΗ"
? "4. ΤΕΛΟΣ"
INPUT "Επιλέξτε [1..4] : " TO CHOICE
DO CASE
   CASE CHOICE=1
      APPEND
   CASE CHOICE=2
      LABEL FORM PELATES
   CASE CHOICE=3
      BROWSE
   OTHERWISE
      QUIT
END CASE
```

Σχ. 6.8. *Η dBASE παρουσιάστηκε στα τέλη της δεκαετίας του 70 από την εταιρία Ashton-Tate αρχικά για μικρουπολογιστές 8-bit και αργότερα για προσωπικούς υπολογιστές. Η dBASE υπήρξε ο σπουδαιότερος εκπρόσωπος εξελιγμένων γλωσσών και εργαλείων προγραμματισμού της εποχής με κύριο χαρακτηριστικό τις εξαιρετικές δυνατότητες διαχείρισης αρχείων (βάσεων) δεδομένων. Η dBASE μπορούσε να λειτουργεί με άμεση εκτέλεση των εντολών της μέσα από μια διαλογική διεπαφή χρήστη, με συνέπεια να μπορεί να χρησιμοποιηθεί και από άτομα με ελάχιστες γνώσεις προγραμματισμού. Το γεγονός αυτό συνέβαλε στην ευρεία διάδοση των προσωπικών υπολογιστών. Η πλέον διαδεδομένη έκδοση ήταν η dBASE III Plus. Εξέλιξή της υπήρξε ο Clipper μεταγλωττιζόμενη γλώσσα, με την οποία δημιουργήθηκαν πλήθος εμπορικών εφαρμογών σε προσωπικούς υπολογιστές. Τα προϊόντα αυτά, που συχνά επονομάζονται xBASE, υπήρξαν προάγγελοι των σημερινών πακέτων διαχείρισης βάσεων δεδομένων όπως η Access. Το απόσπασμα προγράμματος του παραδείγματος παρουσιάζει ένα μενού επιλογών και εκτελεί διακλάδωση στην αντίστοιχη λειτουργία.*

Οι γλώσσες υψηλού επιπέδου χρησιμοποιούν ως εντολές απλές λέξεις της αγγλικής γλώσσας ακολουθώντας αυστηρούς κανόνες σύνταξης, οι οποίες μεταφράζονται από τον ίδιο τον υπολογιστή σε εντολές σε γλώσσα μηχανής

Με τον όρο οπτικό εννοούμε τη δυνατότητα να δημιουργούμε γραφικά ολόκληρο το περιβάλλον της εφαρμογής για παράδειγμα τα πλαίσια διαλόγου ή τα μενού. Με τον όρο οδηγούμενο από το γεγονός εννοούμε τη δυνατότητα να ενεργοποιούνται λειτουργίες του προγράμματος με την εκτέλεση ενός γεγονότος, για παράδειγμα την επιλογή μίας εντολής από ένα μενού ή το κλικ του ποντικιού.

Οι πιο διαδεδομένες γλώσσες προγραμματισμού σε γραφικό περιβάλλον για προσωπικούς υπολογιστές είναι η Visual Basic, η Visual C++ και η Java.

Πλεονεκτήματα των γλωσσών υψηλού επιπέδου

Στα πλεονεκτήματα των γλωσσών προγραμματισμού υψηλού επιπέδου σε σχέση με τις συμβολικές μπορούν να αναφερθούν:

- Ο φυσικότερος και πιο "ανθρώπινος" τρόπος έκφρασης των προβλημάτων. Τα προγράμματα σε γλώσσα υψηλού επιπέδου είναι πιο κοντά στα προβλήματα που επιλύουν.
- Η ανεξαρτησία από τον τύπο του υπολογιστή. Προγράμματα σε μία γλώσσα υψηλού επιπέδου μπορούν να εκτελεστούν σε οποιονδήποτε υπολογιστή με ελάχιστες ή καθόλου μετατροπές. Η δυνατότητα της **μεταφερσιμότητας** των προγραμμάτων είναι σημαντικό προσόν.
- Η ευκολία της εκμάθησης και εκπαίδευσης ως απόρροια των προηγουμένων.
- Η διόρθωση λαθών και η συντήρηση προγραμμάτων σε γλώσσα υψηλού επιπέδου είναι πολύ ευκολότερο έργο.

Συνολικά οι γλώσσες υψηλού επιπέδου ελάττωσαν σημαντικά το χρόνο και το κόστος παραγωγής νέων προγραμμάτων, αφού λιγότεροι προγραμματιστές μπορούν σε μικρότερο χρόνο να αναπτύξουν προγράμματα που χρησιμοποιούνται σε περισσότερους υπολογιστές.

6.2.4 Γλώσσες 4ης γενιάς

Οι γλώσσες υψηλού επιπέδου (γλώσσες 3ης γενιάς) γνώρισαν μεγάλη επιτυχία λόγω των πλεονεκτημάτων που παρουσιάζουν. Ωστόσο απευθύνονται μόνο σε προγραμματιστές. Ο χρήστης ενός υπολογιστή δεν είχε τη δυνατότητα να επιφέρει αλλαγές σε κάποιο πρόγραμμα, προκειμένου να ικανοποιήσει μια νέα ανάγκη του. Σταδιακά όμως πολλές γλώσσες εφοδιάστηκαν με εργαλεία προγραμματισμού που αποκρύπτουν πολλές λεπτομέρειες από τις τεχνικές υλοποίησης και με αυτά ο χρήστης μπορεί να επιλύει μόνος του μικρά προβλήματα εφαρμογών. Αυτή η αυξανόμενη τάση απόκρυψης της αρχιτεκτονικής του υλικού και της τεχνικής του προγραμματισμού οδήγησε στις γλώσσες 4ης γενιάς.

Στις γλώσσες αυτές ο χρήστης ενός υπολογιστή έχει τη δυνατότητα, σχετικά εύκολα, να υποβάλει ερωτήσεις στο σύστημα ή να αναπτύσσει ε-

Ταξινόμηση γλωσσών προγραμματισμού

Όλες οι γλώσσες προγραμματισμού που έχουν αναπτυχθεί μέχρι σήμερα αντιπροσωπεύουν διάφορες ιδέες πάνω στον προγραμματισμό και η κάθε μία είναι συνήθως καλύτερα προσαρμοσμένη σε ορισμένες κατηγορίες προβλημάτων. Η μεγάλη πλειοψηφία των γλωσσών ανήκει στην κατηγορία των **διαδικασιακών** (procedural) γλωσσών. Είναι γνωστές επίσης και ως **αλγοριθμικές** γλώσσες, γιατί είναι σχεδιασμένες για να επιτρέπουν την υλοποίηση αλγορίθμων. Αλλες κατηγορίες γλωσσών υψηλού επιπέδου είναι:

- **Αντικειμενοστραφείς γλώσσες** (object -oriented languages)
- **Συναρτησιακές γλώσσες** (functional languages) π.χ. LISP
- **Μη διαδικασιακές γλώσσες** (non procedural languages) π.χ. PROLOG. Χαρακτηρίζονται επίσης και ως γλώσσες πολύ υψηλού επιπέδου.
- **Γλώσσες ερωταπαντήσεων** (query languages) π.χ. SQL.

Μια άλλη ταξινόμηση μπορεί να προκύψει με βάση την περιοχή χρήσης. Με αυτό το κριτήριο διακρίνουμε:

- **Γλώσσες γενικής χρήσης.** Θεωρητικά κάθε γλώσσα γενικής χρήσης μπορεί να χρησιμοποιηθεί για την επίλυση οποιουδήποτε προβλήματος. Στην πράξη ωστόσο κάθε γλώσσα έχει σχεδιαστεί για να ανταποκρίνεται καλύτερα σε ορισμένη κατηγορία προβλημάτων. Διακρίνονται σε:
  - ✓ **Γλώσσες επιστημονικής κατεύθυνσης** (science-oriented languages) π.χ. FORTRAN
  - ✓ **Γλώσσες εμπορικής κατεύθυνσης** (business-oriented languages) π.χ. COBOL.

  Ας σημειωθεί ότι ορισμένες γλώσσες τα καταφέρνουν εξίσου καλά και στους δύο πρηγούμενους τομείς π.χ. BASIC, Pascal.
- **Γλώσσες προγραμματισμού συστημάτων** (system programming languages) π.χ. C.
- **Γλώσσες τεχνητής νοημοσύνης** (artificial intelligence languages) π.χ. LISP, PROLOG.
- **Γλώσσες ειδικής χρήσης.** Πρόκειται για γλώσσες που χρησιμοποιούνται σε ειδικές περιοχές εφαρμογών όπως π.χ. στα γραφικά με υπολογιστή, στη ρομποτική, στη σχεδίαση ολοκληρωμένων κυκλωμάτων, στα Συστήματα Διοίκησης Βάσεων Δεδομένων, στην εκπαίδευση μέσω υπολογιστή κ.α.

φαρμογές που ανακτούν πληροφορίες από βάσεις δεδομένων και να καθορίζει τον ακριβή τρόπο εμφάνισης αυτών των πληροφοριών, όπως στο παράδειγμα που ακολουθεί.

```
SELECT ENAME, JOB, SAL
   FROM EMPLOYES
   WHERE DEPTNO=20
   AND SAL > 3000;
```

Η ερώτηση αυτή σε SQL εκτελεί αναζήτηση στη βάση δεδομένων EMPLOYES και επιστρέφει το όνομα, τη θέση και το μισθό των υπαλλήλων της διεύθυνσης 20 που κερδίζουν πάνω από 3.000 €.

Ποια είναι η καλύτερη γλώσσα προγραμματισμού

Στην ιστορία του προγραμματισμού έχουν αναπτυχθεί χιλιάδες γλώσσες και αυτή τη στιγμή χρησιμοποιούνται μερικές εκατοντάδες. Υπάρχουν γλώσσες κατάλληλες για ανάπτυξη ειδικών εφαρμογών και άλλες κατάλληλες για γενική χρήση. Υπάρχουν γλώσσες κατάλληλες για εκπαίδευση και άλλες για ανάπτυξη εμπορικών εφαρμογών. Γλώσσες που επιτρέπουν την εύκολη ανάπτυξη εφαρμογών σε γραφικό περιβάλλον και άλλες που εκμεταλλεύονται τα παράλληλα συστήματα. Υπάρχουν γλώσσες πολύ ισχυρές αλλά πολύπλοκες και γλώσσες χωρίς μεγάλες δυνατότητες αλλά απλές και εύκολες στην εκμάθηση. Ο προγραμματιστής καλείται να επιλέξει την "καλύτερη" γλώσσα για να υλοποιήσει το πρόγραμμα.

Μπορούμε να ισχυριστούμε με βεβαιότητα ότι μία γλώσσα προγραμματισμού που να είναι αντικειμενικά καλύτερη από τις άλλες δεν υπάρχει, ούτε πρόκειται να υπάρξει.

Η επιλογή της γλώσσας για την ανάπτυξη μιας εφαρμογής εξαρτάται από το είδος της εφαρμογής, το υπολογιστικό περιβάλλον στο οποίο θα εκτελεστεί, τα προγραμματιστικά περιβάλλοντα που διαθέτουμε και κυρίως τις γνώσεις του προγραμματιστή. Συνήθως ο προγραμματιστής επιλέγει μία γλώσσα, που φυσικά επιτρέπει και διευκολύνει την ανάπτυξη του είδους της εφαρμογής στο συγκεκριμένο περιβάλλον με βάση όμως τις προσωπικές του γνώσεις και προτιμήσεις.

6.3 Φυσικές και τεχνητές γλώσσες

Οι γλώσσες προγραμματισμού αναπτύχθηκαν, για να μπορεί ο προγραμματιστής να δίνει τις εντολές που πρέπει να εκτελέσει ο υπολογιστής. Χρησιμοποιούνται δηλαδή για την επικοινωνία του ανθρώπου και της μηχανής, όπως αντίστοιχα οι φυσικές γλώσσες χρησιμοποιούνται για την επικοινωνία μεταξύ των ανθρώπων. Οι γλώσσες προγραμματισμού, πού είναι τεχνητές γλώσσες, ακολουθούν τις βασικές έννοιες και αρχές της γλωσσολογίας, επιστήμη που μελετά τις φυσικές γλώσσες.

Μία γλώσσα προσδιορίζεται από το αλφάβητό της, το λεξιλόγιό της, τη γραμματική της και τέλος τη σημασιολογία της.

Το αλφάβητο

Αλφάβητο μίας γλώσσας καλείται το σύνολο των στοιχείων που χρησιμοποιείται από τη γλώσσα.

Για παράδειγμα η ελληνική γλώσσα περιέχει τα εξής στοιχεία: Τα γράμματα του αλφαβήτου πεζά και κεφαλαία 48 δηλαδή χαρακτήρες (Α-Ω και α-ω), τα 10 ψηφία (0-9) και όλα τα σημεία στίξης. Αντίστοιχα η αγγλική γλώσσα περιλαμβάνει τα γράμματα του αγγλικού αλφαβήτου (A-Z και a-z) καθώς και τα ψηφία και όλα τα σημεία στίξης που χρησιμοποιούνται.

Το λεξιλόγιο

Το λεξιλόγιο αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου, τις λέξεις που είναι δεκτές από την γλώσσα. Για παράδειγμα στην ελληνική γλώσσα η ακολουθία των γραμμάτων ΑΒΓΑ είναι δεκτή αφού αποτελεί λέξη, αλλά η ακολουθία ΑΒΓΔΑ δεν αποτελεί λέξη της ελληνικής γλώσσας, άρα δεν είναι δεκτή.

Η Γραμματική

Η Γραμματική αποτελείται από το **τυπικό** ή **τυπολογικό** (accidence) και το **συντακτικό** (syntax).

Τυπικό είναι το σύνολο των κανόνων που ορίζει τις μορφές με τις οποίες μία λέξη είναι αποδεκτή. Για παράδειγμα στην ελληνική γλώσσα οι λέξεις γλώσσα, γλώσσας, γλώσσες είναι δεκτές, ενώ η λέξη γλώσσατ δεν είναι αποδεκτή.

Συντακτικό είναι το σύνολο των κανόνων που καθορίζει τη νομιμότητα της διάταξης και της σύνδεσης των λέξεων της γλώσσας για τη δημιουργία προτάσεων.

Η γνώση του συντακτικού επιτρέπει τη δημιουργία σωστών προτάσεων στις φυσικές γλώσσες ενώ στις γλώσσες προγραμματισμού τη δημιουργία σωστών εντολών.

Η σημασιολογία

Κάθε γλώσσα προσδιορίζεται από το αλφάβητο της, το λεξιλόγιο της, τη γραμματική της και τη σημασιολογία της.

Η σημασιολογία (Semantics) είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και κατά επέκταση των εκφράσεων και προτάσεων που χρησιμοποιούνται σε μία γλώσσα.

Στις γλώσσες προγραμματισμού οι οποίες είναι τεχνητές γλώσσες, ο δημιουργός της γλώσσας αποφασίζει τη σημασιολογία των λέξεων της γλώσσας.

Διαφορές φυσικών και τεχνητών γλωσσών.

Μία βασική διαφορά μεταξύ φυσικών και τεχνητών γλωσσών είναι η δυνατότητα εξέλιξής τους. Οι φυσικές γλώσσες εξελίσσονται συνεχώς, νέες λέξεις δημιουργούνται, κανόνες γραμματικής και σύνταξης αλλάζουν με την πάροδο του χρόνου και αυτό γιατί η γλώσσα χρησιμοποιείται για την επικοινωνία μεταξύ ανθρώπων, που εξελίσσονται και αλλάζουν ανάλογα με τις εποχές και τον κοινωνικό περίγυρο.

Αντίθετα οι τεχνητές γλώσσες χαρακτηρίζονται από στασιμότητα, αφού κατασκευάζονται συνειδητά για ένα συγκεκριμένο σκοπό.

Ωστόσο συχνά οι γλώσσες προγραμματισμού βελτιώνονται και μεταβάλονται από τους δημιουργούς τους, με σκοπό να διορθωθούν αδυναμίες ή να καλύψουν μεγαλύτερο εύρος εφαρμογών ή τέλος να ακολουθήσουν τις νέες εξελίξεις. Οι γλώσσες προγραμματισμού αλλάζουν σε επίπεδο διαλέκτου (για παράδειγμα GW-Basic και QuickBasic) ή σε επίπεδο επέκτασης (για παράδειγμα Basic και Visual Basic).

6.4 Τεχνικές σχεδίασης προγραμμάτων

Από την αρχή της εμφάνισης των υπολογιστών γίνονται συνεχείς προσπάθειες ανάπτυξης μεθοδολογιών και τεχνικών προγραμματισμού, που θα εξασφαλίζουν τη δημιουργία απλών και κομψών προγραμμάτων, την

εύκολη γραφή τους όσο και την κατανόησή τους.

6.4.1 Ιεραρχική σχεδίαση προγράμματος

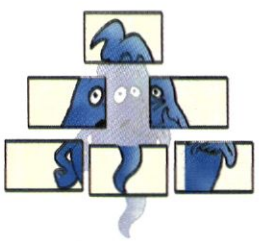
*Η **ιεραρχική σχεδίαση** ή **ιεραρχικός προγραμματισμός** χρησιμοποιεί τη στρατηγική της συνεχούς διαίρεσης του προβλήματος σε υποπροβλήματα*

Η τεχνική της ιεραρχικής σχεδίασης και επίλυσης ή η διαδικασία σχεδίασης "από επάνω προς τα κάτω" όπως συχνά ονομάζεται (top-down program design) περιλαμβάνει τον καθορισμό των βασικών λειτουργιών ενός προγράμματος, σε ανώτερο επίπεδο, και στη συνέχεια τη διάσπαση των λειτουργιών αυτών σε όλο και μικρότερες λειτουργίες, μέχρι το τελευταίο επίπεδο που οι λειτουργίες είναι πολύ απλές, ώστε να επιλυθούν εύκολα.

Σκοπός της ιεραρχικής σχεδίασης είναι η διάσπαση λοιπόν του προβλήματος σε μια σειρά από απλούστερα υποπροβλήματα, τα οποία να είναι εύκολο να επιλυθούν οδηγώντας στην επίλυση του αρχικού προβλήματος.

Για την υποβοήθηση της ιεραρχικής σχεδίασης χρησιμοποιούνται διάφορες διαγραμματικές τεχνικές, όπως για παράδειγμα το διάγραμμα του σχήματος 6.4.

6.4.2 Τμηματικός προγραμματισμός



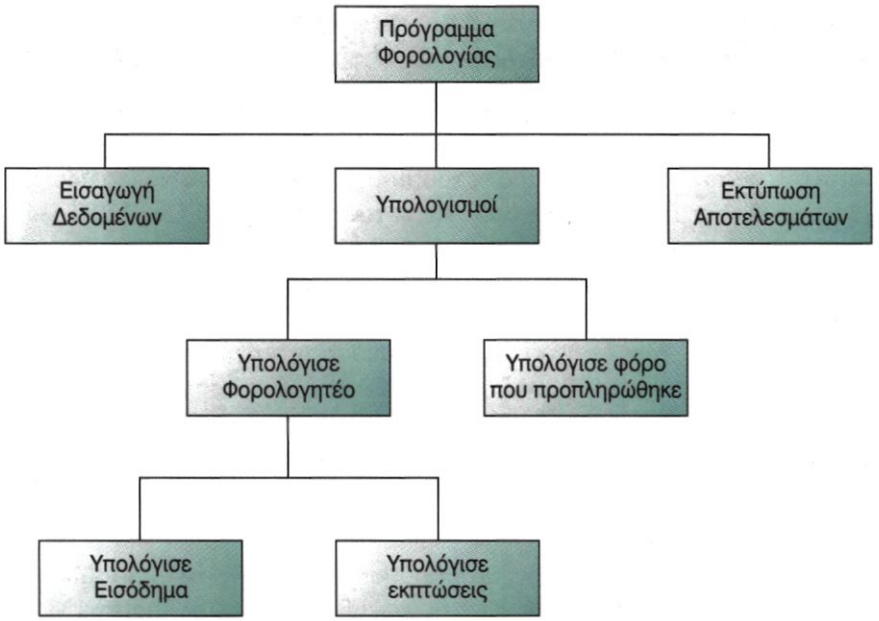
Η ιεραρχική σχεδίαση προγράμματος υλοποιείται με τον τμηματικό προγραμματισμό. Μετά την ανάλυση του προβλήματος σε αντίστοιχα υποπροβλήματα, κάθε υποπρόβλημα αποτελεί ανεξάρτητη **ενότητα** (module), που γράφεται ξεχωριστά από τα υπόλοιπα τμήματα προγράμματος.

Η σωστή διαίρεση του αρχικού προβλήματος σε υποπροβλήματα και κατά συνέπεια του αρχικού προγράμματος σε τμήματα προγράμματος είναι μία διαδικασία αρκετά πολύπλοκη και θα εξεταστεί σε επόμενο κεφάλαιο.

Εδώ πρέπει να σημειωθεί ότι ο τμηματικός προγραμματισμός διευκολύνει τη δημιουργία του προγράμματος, μειώνει τα λάθη και επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.

6.4.3 Δομημένος προγραμματισμός

Η μεθοδολογία που σήμερα έχει επικρατήσει απόλυτα και σχεδόν όλες οι σύγχρονες γλώσσες προγραμματισμού υποστηρίζουν, είναι ο δομημένος

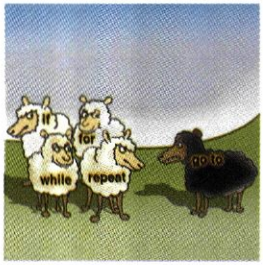


Σχ. 6.4. *Ιεραρχική σχεδίαση υπολογισμού του φόρου εισοδήματος*

προγραμματισμός (structured programming). Ο δομημένος προγραμματισμός παρουσιάστηκε στα μέσα του 1960.

Συγκεκριμένα το 1964 σε ένα συνέδριο στο Ισραήλ παρουσιάστηκε ένα κείμενο των Bohm και Jacopini με τις θεωρητικές αρχές του δομημένου προγραμματισμού. Οι απόψεις τους δεν έγιναν αρχικά ευρύτερα γνωστές και αποδεκτές, αλλά το 1968 ο καθηγητής Edsger Dijkstra δημοσίευσε ένα κείμενο που έκανε ιδιαίτερη αίσθηση και έμελλε να αλλάξει σταδιακά τον τρόπο προγραμματισμού καθώς και τις ίδιες τις γλώσσες προγραμματισμού. Ο τίτλος της μελέτης αυτής ήταν "GO TO Statement Considered Harmful - η εντολή GOTO θεωρείται επιβλαβής" και θεμελίωνε το δομημένο προγραμματισμό. Χρειάστηκε όμως να περάσουν αρκετά χρόνια, ώστε να αρχίσει να διαδίδεται η χρήση του δομημένου προγραμματισμού.

Την εποχή εκείνη δεν υπήρχε μία μεθοδολογία για την ανάπτυξη των προγραμμάτων, τα προγράμματα ήταν μεγάλα και ιδιαίτερα μπερδεμένα με αποτέλεσμα να ξοδεύεται πάρα πολύς χρόνος τόσο στην συγγραφή όσο κύρια στη διόρθωση και τη μετέπειτα συντήρηση τους. Βασικός λόγος για τα προβλήματα αυτά ήταν η αλόγιστη χρήση μίας εντολής, της εντολής GOTO πού χρησιμοποιούμενη άλλαζε διαρκώς τη ροή του προγράμματος. Ο δομημένος προγραμματισμός αναπτύχθηκε από την ανάγκη να υπάρχει



GOTO: Το μαύρο πρόβατο του προγραμματισμού

Στην ιστορία του προγραμματισμού καμία άλλη εντολή δεν συζητήθηκε τόσο πολύ όσο η εντολή **GOTO** (πήγαινε). Η εντολή GOTO έχει ως αποτέλεσμα την αλλαγή της ροής του προγράμματος, της διακλάδωσης σε μία άλλη εντολή του προγράμματος εκτός από την επόμενη. Η εντολή αυτή χώρισε τους προγραμματιστές σε δύο αντιμαχόμενες ομάδες. Η μία αποτελείτο από φανατικούς υποστηρικτές της χρήσης του GOTO, οι οποίοι με τη χρήση της έλυναν εύκολα και αβασάνιστα προβλήματα της ανάπτυξης των προγραμμάτων τους και η δεύτερη με πολέμιους που έβλεπαν ότι η εντολή αυτή ήταν υπεύθυνη για τη δυσκολία στην αρχική σχεδίαση της λύσης, στην παρακολούθηση και κατανόηση του προγράμματος και τέλος στη συντήρηση. Ο δομημένος προγραμματισμός προήλθε από την ανάγκη του περιορισμού της ανεξέλεγκτης χρήσης του GOTO.

Η χρήση της εντολής αυτής θα παρουσιαστεί με ένα απλό παράδειγμα, ενώ για λόγους σύγκρισης δίνεται ο ψευδοκώδικας με χρήση της δομής επιλογής, όπως παρουσιάστηκε στα προηγούμενα κεφάλαια

```
'''''
ΑΝ Αριθμός>0 ΤΟΤΕ GOTO 1
ΑΝ Αριθμός=0 ΤΟΤΕ GOTO 2
 ΓΡΑΨΕ 'Αρνητικός'
GOTO 4
1:ΓΡΑΨΕ 'Θετικός'
GOTO 4
2: ΓΡΑΨΕ 'Μηδέν'
GOTO 4
4:! Συνέχεια,
```

```
''''
ΑΝ Αριθμός>0 ΤΟΤΕ ΓΡΑΨΕ 'Θετικός'
ΑΛΛΙΩΣ_ΑΝ Αριθμός=0  ΤΟΤΕ ΓΡΑΨΕ 'Μηδέν'
ΑΛΛΙΩΣ ΓΡΑΨΕ 'Αρνητικός'
ΤΕΛΟΣ_ΑΝ
'''
```

Η χρήση του GOTO κάνει ακόμα και αυτό το μικρό τμήμα προγράμματος δύσκολο στην κατανόηση του και στην παρακολούθησή του.

⇒

Όλες οι σύγχρονες γλώσσες προγραμματισμού, υποστηρίζουν το δομημένο προγραμματισμό και διαθέτουν εντολές που καθιστούν τη χρήση του GOTO περιττή. Για λόγους όμως συμβατότητας με τις παλιότερες εκδόσεις τους καθώς και για λόγους συντήρησης παλιών προγραμμάτων, μερικές τη διατηρούν στο ρεπερτόριο των εντολών τους.

Στη συνέχεια αυτού του βιβλίου η εντολή GOTO δεν θα μας απασχολήσει και καλό είναι να μη χρησιμοποιείται στην ανάπτυξη προγραμμάτων.

μία κοινή μεθοδολογία στην ανάπτυξη των προγραμμάτων και τη μείωση των εντολών GOTO που χρησιμοποιούνται στο πρόγραμμα.

Ο δομημένος προγραμματισμός δεν είναι απλώς ένα είδος προγραμματισμού, είναι μία μεθοδολογία σύνταξης προγραμμάτων που έχει σκοπό να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων, να μειώσει τα λάθη, να εξασφαλίσει την εύκολη κατανόηση των προγραμμάτων και να διευκολύνει τις διορθώσεις και τις αλλαγές σε αυτά.

Ο δομημένος προγραμματισμός στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή της ακολουθίας, τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα όπως και κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.

Οι τρεις αυτές δομές παρουσιάστηκαν στο κεφάλαιο 2 και θα επαναληφθούν στα επόμενα κεφάλαια.

Αν και ο δομημένος προγραμματισμός αρχικά εμφανίστηκε σαν μία προσπάθεια περιορισμού των εντολών GOTO, σήμερα αποτελεί τη βασική μεθοδολογία προγραμματισμού.

Ο δομημένος προγραμματισμός ενθαρρύνει και βοηθάει την ανάλυση του προγράμματος σε επί μέρους τμήματα, έτσι ώστε σήμερα ο όρος δομημένος προγραμματισμός περιέχει τόσο την ιεραρχική σχεδίαση όσο και τον τμηματικό προγραμματισμό.

Πλεονεκτήματα του δομημένου προγραμματισμού

Επιγραμματικά μπορούμε να αναφέρουμε τα εξής πλεονεκτήματα του δομημένου προγραμματισμού.

- Δημιουργία απλούστερων προγραμμάτων.
- Άμεση μεταφορά των αλγορίθμων σε προγράμματα.
- Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα.
- Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος.
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους.
- Ευκολότερη διόρθωση και συντήρηση.

6.5 Αντικειμενοστραφής προγραμματισμός

Μία νέα ιδέα στον προγραμματισμό γεννήθηκε στις παγωμένες νορβηγικές ακτές στα τέλη της δεκαετίας του '70 και πέρασε πολύ γρήγορα στην άλλη μεριά του Ατλαντικού. Πρόκειται για μια νέα τάση αντιμετώπισης προγραμματιστικών αντιλήψεων και δομών που ονομάζεται **αντικειμενοστραφής** (object-oriented) προγραμματισμός. Την τελευταία δεκαετία έχει γίνει η επικρατούσα κατάσταση και έχει αλλάξει ριζικά τα μέχρι πριν από λίγα χρόνια γνωστά και σταθερά σημεία αναφοράς των προγραμματιστών.

Η ιδέα του αντικειμενοστραφούς προγραμματισμού ή της αντικειμενοστραφούς σχεδίασης έχει τις ρίζες της σε πολύ απλοϊκή ιδέα. Ένα πρόγραμμα περιγράφει "ενέργειες" (επεξεργασία) που εφαρμόζονται πάνω σε δεδομένα. Ένα βασικό ερώτημα που τίθεται είναι αν η φιλοσοφία, η δομή του προγράμματος είναι προτιμότερο να στηρίζεται στις "ενέργειες" ή στα δεδομένα. Η απάντηση σε αυτό το ερώτημα προσδιορίζει και τη βασική διαφορά ανάμεσα στις παραδοσιακές προγραμματιστικές τεχνικές και στην αντικειμενοστραφή προσέγγιση.

Η αντικειμενοστραφής σχεδίαση εκλαμβάνει ως πρωτεύοντα δομικά στοιχεία ενός προγράμματος τα δεδομένα, από τα οποία δημιουργούνται με κατάλληλη μορφοποίηση τα **αντικείμενα** (objects). Αυτή η σχεδίαση αποδείχθηκε ότι επιφέρει καλύτερα αποτελέσματα, αφού τα προγράμματα

που δημιουργούνται είναι περισσότερο ευέλικτα και επαναχρησιμοποιήσιμα. Βέβαια, δημιουργούνται μία σειρά από εύλογα ερωτήματα, όπως "Τι ακριβώς είναι ένα αντικείμενο;", "Πώς προσδιορίζουμε και περιγράφουμε ένα αντικείμενο;", "Πώς το πρόγραμμα χειρίζεται τα αντικείμενα;" "Πώς τα αντικείμενα συσχετίζονται μεταξύ τους;". Απαντήσεις σε αυτά τα ερωτήματα καθώς και αναλυτική παρουσίαση του αντικειμενοστραφούς προγραμματισμού υπάρχουν στο κεφάλαιο 11.

Φυσικά ο αντικειμενοστραφής προγραμματισμός χρησιμοποιεί την ιεραρχική σχεδίαση, τον τμηματικό προγραμματισμό και ακολουθεί τις αρχές του δομημένου προγραμματισμού.

6.6 Παράλληλος προγραμματισμός

Μία άλλη μορφή προγραμματισμού που αναπτύσσεται τελευταία και πιθανόν στο μέλλον να γνωρίσει μεγάλη άνθηση, είναι ο **παράλληλος προγραμματισμός**. Σχετικά πρόσφατα εμφανίστηκαν υπολογιστές που ξεφεύγουν από την κλασική αρχιτεκτονική και διαθέτουν περισσότερους από έναν επεξεργαστές. Οι επεξεργαστές αυτοί μοιράζονται την ίδια μνήμη και λειτουργούν παράλληλα εκτελώντας διαφορετικές εντολές του ιδίου προγράμματος. Οι υπολογιστές αυτοί εμφανίζονται θεωρητικά να πετυχαίνουν ταχύτητες, που είναι ασύλληπτες για τους τυπικούς υπολογιστές με έναν επεξεργαστή. Για να εκμεταλλευτούμε όμως την ταχύτητα που προσφέρει η αρχιτεκτονική τους, πρέπει το πρόβλημα να διαιρεθεί σε τμήματα που εκτελούνται παράλληλα και στη συνέχεια να προγραμματιστεί σε ένα προγραμματιστικό περιβάλλον που να επιτρέπει τον παράλληλο προγραμματισμό.

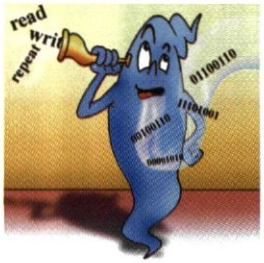
Μια γλώσσα προγραμματισμού που υποστηρίζει παράλληλο προγραμματισμό είναι η OCCAM.

Όπως αναφέρθηκε και στο κεφάλαιο 5, ο παράλληλος προγραμματισμός αποτελεί μία σημαντική επιστημονική περιοχή, η οποία ξεφεύγει από τα όρια αυτού του βιβλίου.

6.7 Προγραμματιστικά περιβάλλοντα

Κάθε πρόγραμμα που γράφτηκε σε οποιαδήποτε γλώσσα προγραμματισμού, πρέπει να μετατραπεί σε μορφή αναγνωρίσιμη και εκτελέσιμη από τον υπολογιστή, δηλαδή σε εντολές γλώσσας μηχανής.

Η μετατροπή αυτή επιτυγχάνεται με τη χρήση ειδικών μεταφραστικών προγραμμάτων. Υπάρχουν δύο μεγάλες κατηγορίες τέτοιων προγραμμά-

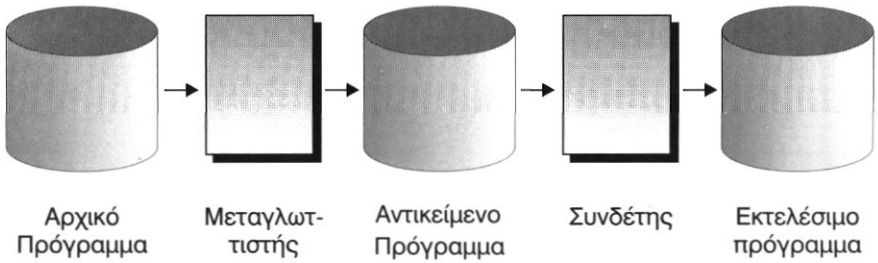


των, οι **μεταγλωττιστές** (compilers) και οι **διερμηνευτές** (interpreters). Ο μεταγλωττιστής δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής. Το τελευταίο μπορεί να εκτελείται οποτεδήποτε από τον υπολογιστή και είναι τελείως ανεξάρτητο από το αρχικό πρόγραμμα. Αντίθετα ο διερμηνευτής διαβάζει μία προς μία τις εντολές του αρχικού προγράμματος και για κάθε μια εκτελεί αμέσως μια ισοδύναμη ακολουθία εντολών μηχανής.

Το αρχικό πρόγραμμα λέγεται **πηγαίο** πρόγραμμα (source), ενώ το πρόγραμμα που παράγεται από το μεταγλωττιστή λέγεται **αντικείμενο** πρόγραμμα (object).

Το αντικείμενο πρόγραμμα είναι μεν σε μορφή κατανοητή από τον υπολογιστή, αλλά συνήθως δεν είναι σε θέση να εκτελεστεί. Χρειάζεται να συμπληρωθεί και να συνδεθεί με άλλα τμήματα προγράμματος απαραίτητα για την εκτέλεσή του, τμήματα που είτε τα γράφει ο προγραμματιστής είτε βρίσκονται στις **βιβλιοθήκες** (libraries) της γλώσσας. Το πρόγραμμα που επιτρέπει αυτή τη σύνδεση ονομάζεται **συνδέτης – φορτωτής** (linker-loader). Το αποτέλεσμα του συνδέτη είναι η παραγωγή του **εκτελέσιμου προγράμματος** (executable), το οποίο είναι το τελικό πρόγραμμα που εκτελείται από τον υπολογιστή. Για το λόγο αυτό η συνολική διαδικασία αποκαλείται μεταγλώττιση και σύνδεση.

Η δημιουργία του εκτελέσιμου προγράμματος γίνεται μόνο στην περίπτωση, που το αρχικό πρόγραμμα δεν περιέχει συντακτικά λάθη. Τις περισσότερες φορές κάθε πρόγραμμα αρχικά θα έχει λάθη. Τα λάθη του προγράμματος είναι γενικά δύο ειδών, λογικά και συντακτικά. Τα λογικά λάθη εμφανίζονται μόνο στην εκτέλεση, ενώ τα συντακτικά λάθη στο στάδιο της μεταγλώττισης.

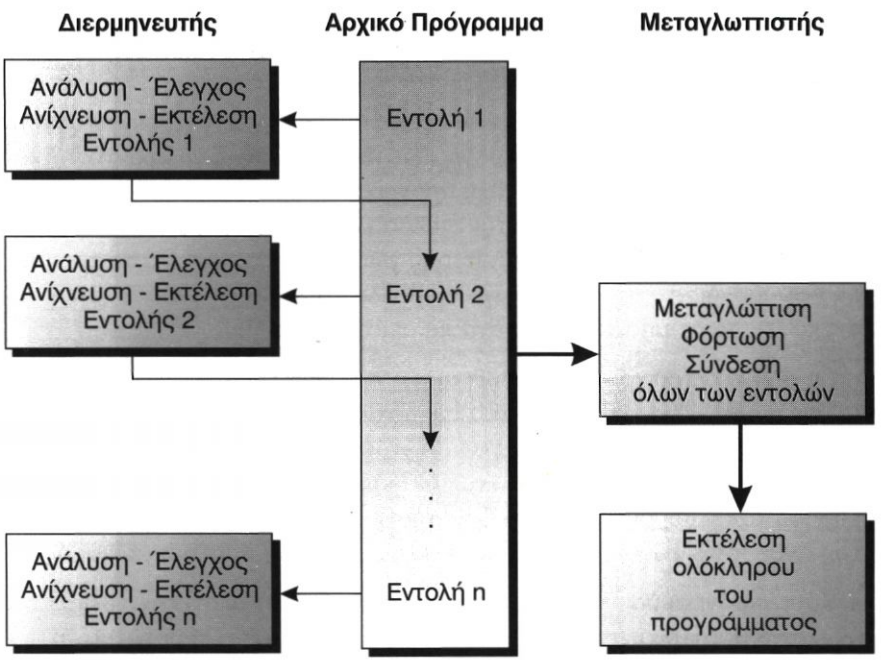


Σχ. 6.5. Μεταγλώττιση και σύνδεση προγράμματος

Εκτενής παρουσίαση των λαθών και των τρόπων που αντιμετωπίζονται γίνεται σε επόμενο κεφάλαιο. Εδώ αναφέρουμε ότι τα λογικά λάθη που είναι τα πλέον σοβαρά και δύσκολα στη διόρθωση τους, οφείλονται σε σφάλματα κατά την υλοποίηση του αλγορίθμου, ενώ τα συντακτικά οφείλονται σε αναγραμματισμούς ονομάτων εντολών, παράληψη δήλωσης δεδομένων και πρέπει πάντα να διορθωθούν, ώστε να παραχθεί το τελικό εκτελέσιμο πρόγραμμα.

Ο μεταγλωττιστής ή ο διερμηνευτής ανιχνεύει λοιπόν τα συντακτικά λάθη και εμφανίζει κατάλληλα διαγνωστικά μηνύματα. Το στάδιο που ακολουθεί είναι η διόρθωση των λαθών. Το διορθωμένο πρόγραμμα επαναϋποβάλεται για μεταγλώττιση και η διαδικασία αυτή επαναλαμβάνεται, μέχρις ότου εξαληφθούν πλήρως όλα τα λάθη.

Η χρήση μεταγλωττιστή έχει το μειονέκτημα, ότι προτού χρησιμοποιηθεί ένα πρόγραμμα, πρέπει να περάσει από τη διαδικασία της μεταγλώττισης και σύνδεσης. Από την άλλη μεριά η χρήση διερμηνευτή έχει το πλεονέκτημα της άμεσης εκτέλεσης και συνεπώς και της άμεσης διόρθωσης. Όμως η εκτέλεση του προγράμματος καθίσταται πιο αργή, σημαντικά μερικές φορές, από εκείνη του ισοδύναμου εκτελέσιμου προγράμματος που παράγει ο μεταγλωττιστής. Πάντως τα σύγχρονα προγραμματιστικά περι-



Σχ. 6.6. Διαδικασία μετάφρασης και εκτέλεσης ενός προγράμματος

Τα σύγχρονα ολοκληρωμένα προγραμματιστικά περιβάλλοντα δεν παρέχουν απλώς ένα μεταφραστή μιας γλώσσας προγραμματισμού. Περιέχουν όλα τα προγράμματα και τα εργαλεία που απαιτούνται και βοηθούν τη συγγραφή, την εκτέλεση και κύρια τη διόρθωση των προγραμμάτων.

βάλλοντα παρουσιάζονται συνήθως με μεικτές υλοποιήσεις, όπου χρησιμοποιείται διερμηνευτής κατά τη φάση δημιουργίας του προγράμματος και μεταγλωττιστής για την τελική έκδοση και εκμετάλλευση του προγράμματος.

Για την αρχική σύνταξη των προγραμμάτων και τη διόρθωσή τους στη συνέχεια χρησιμοποιείται ένα ειδικό πρόγραμμα που ονομάζεται **συντάκτης** (editor). Ο συντάκτης είναι ουσιαστικά ένας μικρός επεξεργαστής κειμένου, με δυνατότητες όμως που διευκολύνουν τη γρήγορη γραφή των εντολών των προγραμμάτων

Για τη δημιουργία, τη μετάφραση και την εκτέλεση ενός προγράμματος απαιτούνται τουλάχιστον τρία προγράμματα: ο συντάκτης, ο μεταγλωττιστής και ο συνδέτης. Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρέχουν αυτά τα προγράμματα με ενιαίο τρόπο.

Το κάθε προγραμματιστικό περιβάλλον έχει φυσικά διαφορετικά εργαλεία και ιδιότητες. Για παράδειγμα ένα περιβάλλον οπτικού (visual) προγραμματισμού πρέπει να περιέχει οπωσδήποτε και ειδικό συντάκτη που να διευκολύνει τη δημιουργία γραφικών αντικειμένων (για παράδειγμα φόρμες, λίστες, παράθυρα διαλόγου) παρέχοντας στον προγραμματιστή τα αντίστοιχα γραφικά εργαλεία.

Ανακεφαλαίωση

Δημιουργία προγράμματος είναι η μετατροπή του αλγορίθμου που επιλύει ένα πρόβλημα σε εντολές προγράμματος. Οι εντολές γράφονται σε κάποια από τις εκατοντάδες γλώσσες προγραμματισμού, που έχουν αναπτυχθεί με σκοπό να διευκολύνουν την επίλυση συγκεκριμένου τύπου προβλημάτων. Η επιλογή της καταλληλότερης γλώσσας εξαρτάται από το είδος της εφαρμογής, το υπολογιστικό περιβάλλον που θα εκτελεστεί, τις γνώσεις και τις προτιμήσεις του προγραμματιστή.

Οι τεχνικές που χρησιμοποιούνται για την ανάπτυξη προγραμμάτων είναι της ιεραρχικής σχεδίασης, του τμηματικού προγραμματισμού και του δομημένου προγραμματισμού, που επιτρέπουν τη δημιουργία προγραμμάτων κατανοητών και απλών, ενώ διευκολύνουν τη διόρθωση και τη συντήρηση των εφαρμογών. Ένα είδος προγραμματισμού που γνωρίζει ιδιαίτερη άνθηση τελευταία, είναι ο αντικειμενοστραφής προγραμματισμός.

Κάθε πρόγραμμα για να εκτελεστεί από τον υπολογιστή χρειάζεται πρώτα να μετατραπεί σε μορφή κατανοητή από αυτόν. Η μετατροπή αυτή γίνεται από τους μεταγλωττιστές ή τους διερμηνευτές, οι οποίοι επιση-

μαίνουν και τα συντακτικά λάθη, που έχει κάθε πρόγραμμα. Η σύνταξη του προγράμματος, η μετάφραση, η διόρθωση των λαθών και η εκτέλεση του γίνεται με τα ολοκληρωμένα προγραμματιστικά περιβάλλοντα που διαθέτουν πολλά εργαλεία για την υποβοήθηση της ανάπτυξης των εφαρμογών.

Ερωτήσεις - Θέματα για συζήτηση

- Τι ονομάζεται πρόγραμμα;
- Τι είναι οι γλώσσες μηχανής;
- Ποιες οι διαφορές των γλωσσών υψηλού επιπέδου από αυτές χαμηλού επιπέδου;
- Ποιες γλώσσες υψηλού επιπέδου γνωρίζεις;
- Τι ονομάζουμε οπτικό προγραμματισμό και τι οδηγούμενο από τα γεγονότα;
- Πώς προσδιορίζεται μία φυσική γλώσσα;
- Ποιες οι κυριότερες διαφορές των φυσικών και των τεχνητών γλωσσών;
- Πώς γίνεται η παράσταση της ιεραρχικής σχεδίασης προγράμματος;
- Ποιες οι αρχές του δομημένου προγραμματισμού;
- Ποια τα πλεονεκτήματα του δομημένου προγραμματισμού;
- Τι ονομάζεται αντικειμενοστραφής προγραμματισμός;
- Ποια η διαδικασία για την μετάφραση και εκτέλεση ενός προγράμματος;
- Ποιες οι διαφορές μεταγλωττιστή και διερμηνευτή;
- Ποια προγράμματα και εργαλεία περιέχει ένα προγραμματιστικό περιβάλλον;

Λέξεις κλειδιά

Πρόγραμμα, Γλώσσα μηχανής, Συμβολική γλώσσα, Γλώσσες υψηλού επιπέδου, Τμηματικός προγραμματισμός, Δομημένος προγραμματισμός, Αντικειμενοστραφής προγραμματισμός, Μεταγλωττιστής, Διερμηνευτής, Προγραμματιστικό περιβάλλον

Βιβλιογραφία

1. Ph. Breton, *Ιστορία της Πληροφορικής*, Εκδόσεις Δίαυλος, Αθήνα.

2. Γ. Μπαμπινιώτης, *Θεωρητική Γλωσσολογία*, Αθήνα, 1986.

3. Χρ. Κοίλιας-Στρ., Καλαφατούδης, *Το πρώτο βιβλίο της Πληροφορικής*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.

4. *Εγκυκλοπαίδεια Πληροφορικής και Τεχνολογίας Υπολογιστών*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1986.

5. Αθ. Τσουροπλής-Στ. Κλημόπουλος, *Από τη FORTRAN 77 στη FORTRAN 90*, Εκδόσεις Πελεκάνος, Αθήνα, 1995.

6. Χ. Κοίλας-Στρ. Μαραγκός, *Η γλώσσα COBOL και οι εφαρμογές της*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.

7. Κ. Μαρινάκης-Ν. Ιωαννίδης, *Structure & Advanced COBOL*, Εκδόσεις Έλιξ, Αθήνα, 1992.

8. Χ. Κοίλιας-Αλ. Τομαράς, *GWBASIC Θεωρία και Εφαρμογές*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.

9. Μ. Κατζουράκη-Μ. Γεργατσούλης-Σ. Κόκκοτος, *PROγραμματίζοντας στη LOGική*, Έκδοση ΕΠΥ, Αθήνα, 1991.

10. Αικ. Γεωργόπουλου, *LOGO Βήμα προς Βήμα*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1991.

11. Αλ. Τομαράς, *C Θεωρία και Πράξη*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1995.

12. Μ. Μαλιάππης, *SQL Περιβάλλοντα Ανάπτυξης Εφαρμογών 4ης Γενιάς*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1995.

13. E. Horowitz, *Βασικές αρχές γλωσσών προγραμματισμού*, Κλειδάριθμος, Αθήνα, 1995.

14. R. Shackelford, *Introduction to Computing and Algorithms*, Addison-Wesley, USA, 1998.

15. W. Hutching-H Somers, *An Introduction to Machine Translation*, Academic Press, London, 1992.

ΚΕΦΑΛΑΙΟ 7ο

ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

ΠΕΡΙΕΧΟΜΕΝΑ

- Αλφάβητο και τύποι δεδομένων
- Σταθερές και μεταβλητές
- Τελεστές, συναρτήσεις και εκφράσεις
- Εντολή εκχώρησης
- Εντολές εισόδου - εξόδου
- Δομή προγράμματος

Εισαγωγή

Κάθε γλώσσα προγραμματισμού, όπως αναφέρθηκε, έχει το δικό της λεξιλόγιο και τα προγράμματα της ακολουθούν αυστηρούς γραμματικούς και συντακτικούς κανόνες. Για τη δημιουργία σωστών προγραμμάτων είναι απαραίτητη η γνώση των εντολών και του τρόπου σύνταξής τους.

Στο κεφάλαιο αυτό θα παρουσιαστούν τα βασικά στοιχεία της **ΓΛΩΣΣΑΣ**. Θα ασχοληθούμε με τους τύπους δεδομένων που υποστηρίζει, τα είδη των μεταβλητών της, τον τρόπο που υπολογίζονται οι παραστάσεις καθώς και τη δομή που πρέπει να ακολουθεί κάθε πρόγραμμα.

Επίσης θα παρουσιαστούν οι βασικές εντολές της **ΓΛΩΣΣΑΣ**, η εντολή εκχώρησης τιμών σε μεταβλητές και οι εντολές εισόδου εξόδου, με τις οποίες το πρόγραμμα επικοινωνεί με το χρήστη.

Διδακτικοί στόχοι

Να είναι σε θέση ο μαθητής:

- Να διακρίνει τις σταθερές από τις μεταβλητές.
- Να αναγνωρίζει τους διάφορους τύπους μεταβλητών.
- Να μετατρέπει τις αριθμητικές πράξεις σε εντολές προγράμματος.
- Να διατυπώνει τη δομή ενός προγράμματος.
- Να συντάσσει απλά προγράμματα, τα οποία εισάγουν δεδομένα, τα επεξεργάζονται και εμφανίζουν τα αποτελέσματα στην οθόνη.

Προερωτήσεις

- ✓ Έχει ένα πρόγραμμα συγκεκριμένους κανόνες στον τρόπο που γράφεται;
- ✓ Πώς διαχειρίζεται ένα πρόγραμμα τα δεδομένα;
- ✓ Πώς εκτελούνται οι πράξεις σε ένα πρόγραμμα;
- ✓ Με ποιο τρόπο επικοινωνεί το πρόγραμμα με το χρήστη κατά την εκτέλεση του;

Εκατοντάδες γλώσσες προγραμματισμού χρησιμοποιούνται όπως αναφέρθηκε σήμερα για την επίλυση των προβλημάτων με τον υπολογιστή, τη δημιουργία σωστών προγραμμάτων. Η επιλογή της κατάλληλής γλώσσας δεν είναι εύκολη και εξαρτάται από το είδος του προγράμματος, το διαθέσιμο εξοπλισμό και σαφώς τις γνώσεις και τις ιδιαίτερες προτιμήσεις του προγραμματιστή. Συχνά το ίδιο πρόβλημα μπορεί να λυθεί εξίσου ικανοποιητικά με πολλές διαφορετικές γλώσσες προγραμματισμού.

Πρέπει να έχουμε πάντα υπόψη μας ότι:

- Κάθε γλώσσα προγραμματισμού σχεδιάζεται για συγκεκριμένο σκοπό, δίνοντας ιδιαίτερη έμφαση σε ορισμένα χαρακτηριστικά σε βάρος βέβαια κάποιων άλλων. Δεν υπάρχει καλύτερη γλώσσα προγραμματισμού, απλά υπάρχει γλώσσα καταλληλότερη για την ανάπτυξη συγκεκριμένου τύπου εφαρμογών.
- Οι γλώσσες προγραμματισμού περιέχουν πολλές πληροφορίες που σχετίζονται με τεχνικά θέματα. Αυτά τα χαρακτηριστικά αλλάζουν αρκετά συχνά, όπως η γλώσσα εξελίσσεται και εξαρτώνται σε μεγάλο βαθμό από τον εξοπλισμό και το λειτουργικό σύστημα. Οι νεώτερες εκδόσεις των γλωσσών συνήθως διαθέτουν πλουσιότερο ρεπερτόριο εντολών και άλλων δυνατοτήτων, χωρίς όμως να προσθέτουν οτιδήποτε στην εκμάθηση της δημιουργίας σωστών προγραμμάτων.
- Σχεδόν όλες οι γλώσσες προγραμματισμού έχουν κοινά χαρακτηριστικά, επεξεργάζονται κατά κανόνα τους ίδιους τύπους δεδομένων, υποστηρίζουν τις ίδιες βασικές δομές και έχουν παρόμοιες εντολές.

Η γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε στα επόμενα κεφάλαια που ονομάζεται **ΓΛΩΣΣΑ**, είναι σχεδιασμένη έτσι ώστε να αποτελέσει ένα εργαλείο προγραμματισμού κατάλληλο για εκπαιδευτικούς σκοπούς. Περιέχει τα χαρακτηριστικά, τις δομές και τις εντολές που περιέχονται σε διάφορες σύγχρονες γλώσσες προγραμματισμού όπως η Pascal, Visual Basic, C, C++, Java και άλλες, χωρίς όμως να ασχολείται με τις τεχνικές λεπτομέρειες αυτών.

Έτσι ο προγραμματισμός με τη **ΓΛΩΣΣΑ** εστιάζεται στην ανάπτυξη του αλγορίθμου και τη μετατροπή του σε σωστό πρόγραμμα.

Σε όλο το βιβλίο οι εντολές της **ΓΛΩΣΣΑΣ** είναι γραμμένες με μπλε χρώμα και είναι πάντα με κεφαλαία, ενώ οι μεταβλητές είναι με πεζά ή κεφαλαία αλλά με το πρώτο γράμμα πάντα κεφαλαίο. Τα σχόλια των προγραμμάτων είναι γραμμένα με πράσινο χρώμα.

7.1 Το αλφάβητο της ΓΛΩΣΣΑΣ

Το αλφάβητο της **ΓΛΩΣΣΑΣ** αποτελείται από τα γράμματα του ελληνικού και του λατινικού αλφαβήτου, τα ψηφία, καθώς και από ειδικά σύμβολα, που χρησιμοποιούνται για προκαθορισμένες ενέργειες, στις οποίες θα αναφερθούμε στη συνέχεια.

Συγκεκριμένα

Γράμματα

Κεφαλαία ελληνικού αλφαβήτου (Α-Ω)

Πεζά ελληνικού αλφαβήτου (α-ω)

Κεφαλαία λατινικού αλφαβήτου (A-Z)

Πεζά λατινικού αλφαβήτου (a-z)

Ψηφία

0-9

Ειδικοί χαρακτήρες

+ – */= · ( ) . , ' ! & κενός χαρακτήρας ^

7.2 Τύποι δεδομένων

Οι υπολογιστές επεξεργάζονται δεδομένα διαφόρων τύπων, γι αυτό είναι σημαντικό να κατανοήσουμε τους διαφορετικούς τύπους δεδομένων που χειρίζεται η **ΓΛΩΣΣΑ**.

Οι τύποι δεδομένων που υποστηρίζει η **ΓΛΩΣΣΑ** είναι οι αριθμητικοί, που περιλαμβάνουν τους ακέραιους και τους πραγματικούς αριθμούς, οι χαρακτήρες και τέλος οι λογικοί.

Ακέραιος τύπος. Ο τύπος αυτός περιλαμβάνει τους ακέραιους που είναι γνωστοί από τα μαθηματικά. Οι ακέραιοι μπορούν να είναι θετικοί, αρνητικοί ή μηδέν. Παραδείγματα ακεραίων είναι οι αριθμοί 1, 3409, 0, -980.

Πραγματικός τύπος. Ο τύπος αυτός περιλαμβάνει τους πραγματικούς αριθμούς που γνωρίζουμε από τα μαθηματικά. Οι αριθμοί 3.14159,

2.71828, -112.45, 0.45 είναι πραγματικοί αριθμοί. Και οι πραγματικοί αριθμοί μπορούν να είναι θετικοί, αρνητικοί ή μηδέν.

Χαρακτήρας. Ο τύπος αυτός αναφέρεται τόσο σε ένα χαρακτήρα όσο και μία σειρά χαρακτήρων. Τα δεδομένα αυτού του τύπου μπορούν να περιέχουν οποιοδήποτε χαρακτήρα παράγεται από το πληκτρολόγιο. Παραδείγματα χαρακτήρων είναι 'Κ', 'Κώστας', 'σήμερα είναι Τετάρτη', 'Τα πολλαπλάσια του 15 είναι'.

Οι χαρακτήρες πρέπει υποχρεωτικά να βρίσκονται μέσα σε απλά εισαγωγικά, ' '. Τα δεδομένα αυτού του τύπου, επειδή περιέχουν τόσο αλφαβητικούς όσο και αριθμητικούς χαρακτήρες, ονομάζονται συχνά **αλφαριθμητικά**.

Λογικός. Αυτός ο τύπος δέχεται μόνο δύο τιμές ΑΛΗΘΗΣ και ΨΕΥΔΗΣ. Οι τιμές αντιπροσωπεύουν αληθείς ή ψευδείς συνθήκες.

Στην πραγματικότητα τα δεδομένα καταχωρούνται στη μνήμη του υπολογιστή καταλαμβάνοντας συγκεκριμένο αριθμό θέσεων (bytes). Ανάλογα με τον τύπο του δεδομένου και το διατιθέμενο αριθμό bytes ποικίλει και το εύρος τιμών που μπορούν να λάβουν. Έτσι στον υπολογιστή διαθέτουμε ένα υποσύνολο ακεραίων ή πραγματικών αριθμών. Συνήθεις τύποι δεδομένων στις διάφορες γλώσσες προγραμματισμού είναι ο ακέραιος (integer) σε 1, 2 ή 4 bytes και ο πραγματικός (real) σε 4 ή 8 bytes.

7.3 Σταθερές

Οι σταθερές (constants) είναι προκαθορισμένες τιμές που δεν μεταβάλλονται κατά τη διάρκεια εκτέλεσης του προγράμματος. Οι σταθερές είναι αντίστοιχου τύπου δεδομένων, δηλαδή ακέραιες, πραγματικές, αλφαριθμητικές ή λογικές.

Συμβολικές σταθερές

Η **ΓΛΩΣΣΑ** επιτρέπει την αντιστοίχιση σταθερών τιμών με ονόματα, εφόσον αυτά δηλωθούν στην αρχή του προγράμματος (στο τμήμα δήλωσης σταθερών, βλέπε παρακάτω).

Σύνταξη

```
ΣΤΑΘΕΡΕΣ
    Όνομα-1 = σταθερή-τιμή-1
    Όνομα-2 = σταθερά-τιμή-2
    .
    .
    .
    Όνομα-ν = σταθερά-τιμή-ν
```

Παραδείγματα

```
ΣΤΑΘΕΡΕΣ
    ΠΙ=3.14159
    ΦΠΑ=0.19
    ΟΝΟΜΑ='Κώστας'
```

Λειτουργία

Αποδίδει ονόματα σε σταθερές τιμές. Κάθε ένα από αυτά τα ονόματα μπορεί να χρησιμοποιηθεί οπουδήποτε στο πρόγραμμα, αλλά δεν είναι δυνατή η μεταβολή της τιμής κατά τη διάρκεια εκτέλεσης του προγράμματος.

Η χρήση ονομάτων σταθερών κάνει το πρόγραμμα πιο κατανοητό και κατά συνέπεια ευκολότερο να διορθωθεί και να συντηρηθεί.

Ονόματα

Κάθε πρόγραμμα, καθώς και τα δεδομένα που χρησιμοποιεί (συμβολικές σταθερές και μεταβλητές) έχουν ένα όνομα, με το οποίο αναφερόμαστε σε αυτά. Τα ονόματα αυτά μπορούν να αποτελούνται από γράμματα πεζά ή κεφαλαία του ελληνικού ή του λατινικού αλφαβήτου (Α-Ω, A-Z), ψηφία (0-9) καθώς και τον χαρακτήρα κάτω παύλα (underscore) (_), ενώ πρέπει υποχρεωτικά να αρχίζουν με γράμμα.

Επειδή μερικές λέξεις χρησιμοποιούνται από την ίδια τη **ΓΛΩΣΣΑ** για συγκεκριμένους λόγους, όπως οι λέξεις ΠΡΟΓΡΑΜΜΑ, ΑΚΕΡΑΙΕΣ, ΠΡΑΓΜΑΤΙΚΕΣ, ΑΝ κ.λπ, αυτές οι λέξεις δεν μπορούν να χρησιμοποιηθούν ως ονόματα. Οι λέξεις αυτές αποκαλούνται δεσμευμένες.

Παραδείγματα ονομάτων που είναι αποδεκτά από τη **ΓΛΩΣΣΑ** είναι: Α, Όνομα, Τιμή, Τυπική_Απόκλιση, Α100, ΦΠΑ, μέγιστο, Υπολογισμός_Ταχύτητας.

Παραδείγματα ονομάτων που δεν είναι αποδεκτά είναι: 100Α, Μέση Τιμή, Κόστος$.

7.4 Μεταβλητές

Η έννοια της μεταβλητής (variable) είναι γνωστή από τα μαθηματικά. Για παράδειγμα ο τύπος της γεωμετρίας

$$E=\alpha\beta$$

υπολογίζει το εμβαδόν (**E**) ενός ορθογωνίου με διαστάσεις, που συμβολίζονται με **α** και **β**. Αν στο **α** και στο **β** δοθούν οι αντίστοιχες τιμές, τότε ο τύπος αυτός υπολογίζει το εμβαδόν του ορθογωνίου.

Μια μεταβλητή λοιπόν, παριστάνει μία ποσότητα που η τιμή της μπορεί να μεταβάλλεται.

Οι μεταβλητές που χρησιμοποιούνται σε ένα πρόγραμμα, αντιστοιχούνται από το μεταγλωττιστή σε συγκεκριμένες θέσεις μνήμης του υπολογιστή. Η τιμή της μεταβλητής είναι η τιμή που βρίσκεται στην αντίστοιχη θέση μνήμης και όπως αναφέρθηκε μπορεί να μεταβάλλεται κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Μπορούμε να παρομοιάσουμε τη μεταβλητή και την αντίστοιχη θέση μνήμης σαν ένα γραμματοκιβώτιο, το οποίο εξωτερικά έχει ως όνομα το όνομα της μεταβλητής και ως περιεχόμενο εσωτερικά, την τιμή που έχει εκείνη τη συγκεκριμένη στιγμή η μεταβλητή.

Ενώ η τιμή της μεταβλητής μπορεί να αλλάζει κατά την εκτέλεση του προγράμματος, αυτό που μένει υποχρεωτικά αναλλοίωτο είναι ο τύπος της μεταβλητής.

Η **ΓΛΩΣΣΑ** επιτρέπει τη χρήση μεταβλητών των τεσσάρων τύπων που αναφέρθηκαν, δηλαδή ακεραίων, πραγματικών, χαρακτήρων και λογικών ενώ η δήλωση του τύπου κάθε μεταβλητής γίνεται υποχρεωτικά στο τμήμα δήλωσης μεταβλητών.

Το όνομα κάθε μεταβλητής, ακολουθεί τους κανόνες δημιουργίας ονομάτων, δηλαδή αποτελείται από γράμματα, ψηφία καθώς και τον χαρακτήρα _, ενώ το όνομα κάθε μεταβλητής είναι μοναδικό για κάθε πρόγραμμα.

Σύνταξη

```
ΜΕΤΑΒΛΗΤΕΣ
    τύπος-1: Λίστα-μεταβλητών-1
    τύπος-2: Λίστα-μεταβλητών-2
    .
    .
    .
    Τύπος-ν: Λίστα-μεταβλητών-ν
```

Παραδείγματα

```
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ: Εμβαδόν, Α
    ΑΚΕΡΑΙΕΣ: ΤΙΜΗ, Ν
    ΧΑΡΑΚΤΗΡΕΣ: Όνομα
    ΛΟΓΙΚΕΣ: Έλεγχος
```

Λειτουργία

Δηλώνει τον τύπο όλων των μεταβλητών που χρησιμοποιούνται στο πρόγραμμα.

Συνιστάται τα ονόματα των μεταβλητών και των σταθερών να ανάγουν στο περιεχόμενο τους.

Αν και όπως αναφέρθηκε, το όνομα των μεταβλητών μπορεί να είναι οποιοσδήποτε συνδυασμός χαρακτήρων, είναι καλή πρακτική να χρησιμοποιούνται ονόματα, τα οποία να υπονοούν το περιεχόμενό τους, κάνοντας το πρόγραμμα ευκολότερο στην ανάγνωση του και στην κατανόηση του.

Για παράδειγμα στην περίπτωση του υπολογισμού του εμβαδού είναι προτιμότερη η χρήση του ονόματος ΕΜΒΑΔΟ για την αντίστοιχη μεταβλητή, από ένα όνομα που αποτελείται από ένα μόνο γράμμα όπως Ε ή Α ή ένα οποιοδήποτε τυχαίο όνομα που δεν ανάγει στο πραγματικό περιεχόμενο της μεταβλητής όπως Τιμή.

7.5 Αριθμητικοί τελεστές

Οι αριθμητικοί τελεστές που υποστηρίζονται από τη **ΓΛΩΣΣΑ** καλύπτουν τις βασικές πράξεις: πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση ενώ υποστηρίζεται και η ύψωση σε δύναμη, η ακέραια διαίρεση και το υπόλοιπο της ακέραιας διαίρεσης.

Οι τελεστές και οι αντίστοιχες πράξεις είναι:

Αριθμητικός τελεστής	Πράξη
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
^	Ύψωση σε δύναμη
DIV	Ακέραια διαίρεση
MOD	Υπόλοιπο ακέραιας διαίρεσης

Ο τελεστής div χρησιμοποιείται για τον υπολογισμό του πηλίκου μιας διαίρεσης ακεραίων αριθμών, ενώ ο τελεστής mod για το υπόλοιπο. Π.χ.

7 div 2 = 3 και 7 mod 2 = 1

7.6 Συναρτήσεις

Πολλές γνωστές συναρτήσεις από τα μαθηματικά χρησιμοποιούνται συχνά και περιέχονται στη **ΓΛΩΣΣΑ**. Οι συναρτήσεις αυτές είναι:

ΗΜ(Χ)	Υπολογισμός ημιτόνου
ΣΥΝ(Χ)	Υπολογισμός συνημιτόνου
ΕΦ(Χ)	Υπολογισμός εφαπτομένης
Τ_Ρ(Χ)	Υπολογισμός τετραγωνικής ρίζας
ΛΟΓ(Χ)	Υπολογισμός φυσικού λογαρίθμου
Ε(Χ)	Υπολογισμός του e^x
Α_Μ(Χ)	Ακέραιο μέρος του Χ
Α_Τ(Χ)	Απόλυτη τιμή του Χ

7.7 Αριθμητικές εκφράσεις

Οταν μια τιμή προκύπτει από υπολογισμό, τότε αναφερόμαστε σε **εκφράσεις** (expressions). Για τη σύνταξη μιας αριθμητικής έκφρασης χρησιμοποιούνται αριθμητικές σταθερές, μεταβλητές, συναρτήσεις, αριθμητικοί τελεστές και παρενθέσεις. Οι αριθμητικές εκφράσεις υλοποιούν απλές ή σύνθετες μαθηματικές πράξεις.

Κάθε έκφραση παριστάνει μια συγκεκριμένη αριθμητική τιμή, η οποία βρίσκεται μετά την εκτέλεση των πράξεων. Γι' αυτό είναι απαραίτητο όλες

οι μεταβλητές, που εμφανίζονται σε μια έκφραση να έχουν οριστεί προηγούμενα, δηλαδή να έχουν κάποια τιμή.

Ιεραρχία

Οι πράξεις που παρουσιάζονται σε μια έκφραση, εκτελούνται σύμφωνα με την επόμενη ιεραρχία

1. Ύψωση σε δύναμη
2. Πολλαπλασιασμός και διαίρεση
3. Πρόσθεση και αφαίρεση

Παραδείγματα

Μαθηματικά	ΓΛΩΣΣΑ
a+1	a+1
$1/2\ a^3$	1/2*a^3
$\frac{3x+2y}{a-b}$	(3*x+2*y)/(a-b)
2ημχ	2*ΗΜ(χ)

Πάντα πρέπει να χρησιμοποιούνται ζεύγη παρενθέσεων. Διαφορετικός αριθμός αριστερών από δεξιές παρενθέσεις στην ίδια έκφραση είναι ένα από τα πιο συνηθισμένα λάθη.

Οταν η ιεραρχία είναι ίδια, τότε οι πράξεις εκτελούνται από τ' αριστερά προς τα δεξιά. Σε πολλές όμως περιπτώσεις είναι απαραίτητο να προηγηθεί μια πράξη χαμηλότερης ιεραρχίας. Αυτό επιτυγχάνεται με την εισαγωγή των παρενθέσεων. Η πράξη που πρέπει να προηγηθεί περικλείεται σε ένα ζεύγος παρενθέσων, οπότε και εκτελείται πρώτη. Π.χ. η έκφραση 2+3*4 δίδει ως αποτέλεσμα 14, ενώ η (2+3)*4 δίδει 20, διότι εκτελείται πρώτα η πρόσθεση και μετά ο πολλαπλασιασμός.

7.8 Εντολή εκχώρησης

Η εντολή εκχώρησης χρησιμοποιείται για την απόδοση τιμών στις μεταβλητές κατά τη διάρκεια εκτέλεσης του προγράμματος.

Σύνταξη

```
Όνομα-Μεταβλητής <- έκφραση
```

Παραδείγματα

```
A <- 132
ΜΗΝΑΣ <- 'Ιανουάριος'
ΕΜΒΑΔΟΝ <- A*B
```

Λειτουργία

Υπολογίζεται η τιμή της έκφρασης στη δεξιά πλευρά και εκχωρείται η τιμή αυτή στη μεταβλητή, που αναφέρεται στην αριστερή πλευρά.

Μια εντολή εκχώρησης σε καμμία περίπτωση δεν πρέπει να εκλαμβάνεται ως εξίσωση. Στην εξίσωση το αριστερό μέλος ισούται με το δεξιό, ενώ στην εντολή εκχώρησης η τιμή του δεξιού μέλους εκχωρείται, μεταβιβάζεται, αποδίδεται στη μεταβλητή του αριστερού μέλους. Για το λόγο αυτό ως τελεστής εκχώρησης χρησιμοποιείται το σύμβολο <– προκειμένου να διαφοροποιείται από το ίσον (=). Ωστόσο, ας σημειωθεί, ότι οι διάφορες γλώσσες προγραμματισμού χρησιμοποιούν διάφορετικά σύμβολα για το σκοπό αυτό.

Σε μια εντολή εκχώρησης η μεταβλητή και η έκφραση πρέπει να είναι του ιδίου τύπου.

7.9 Εντολές εισόδου-εξόδου

Σχεδόν όλα τα προγράμματα υπολογιστή δέχονται κάποια δεδομένα, τα επεξεργάζονται, υπολογίζουν τα αποτελέσματα και τέλος τα εμφανίζουν.

Τα δεδομένα εισάγονται κατά τη διάρκεια της εκτέλεσης του προγράμματος από μία μονάδα εισόδου, για παράδειγμα το πληκτρολόγιο και τα αποτελέσματα γράφονται σε μία μονάδα εξόδου, για παράδειγμα την οθόνη.

Η **ΓΛΩΣΣΑ** υποστηρίζει για την εισαγωγή δεδομένων από το πληκτρολόγιο την εντολή **ΔΙΑΒΑΣΕ** και για την εμφάνιση των αποτελεσμάτων την εντολή **ΓΡΑΨΕ**.

Σύνταξη

ΔΙΑΒΑΣΕ λίστα-μεταβλητών

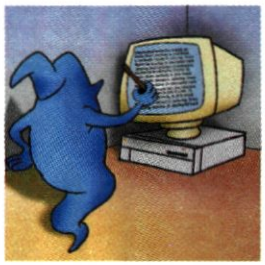
Παραδείγματα

ΔΙΑΒΑΣΕ Ποσότητα, Τιμή

Λειτουργία

Η εκτέλεση της εντολής οδηγεί στην είσοδο τιμών από το πληκτρολόγιο και την εκχώρηση τους στις μεταβλητές που αναφέρονται.

Η εντολή **ΔΙΑΒΑΣΕ** ακολουθείται πάντοτε από ένα ή περισσότερα ονόματα μεταβλητών. Αν υπάρχουν περισσότερες από μία μεταβλητές τότε αυτές χωρίζονται με κόμμα (,). Κατά την εκτέλεση του προγράμματος η εντολή **ΔΙΑΒΑΣΕ** διακόπτει την εκτέλεσή του και το πρόγραμμα περιμένει την εισαγωγή από το πληκτρολόγιο τιμών, που θα εκχωρηθούν στις μεταβλητές. Μετά την ολοκλήρωση της εντολής η εκτέλεση του προγράμματος συνεχίζεται με την επόμενη εντολή.



Σύνταξη

ΓΡΑΨΕ λίστα-στοιχείων

Παραδείγματα

ΓΡΑΨΕ 'Η τετραγωνική ρίζα του', Α,' είναι: ',ΡΙΖΑ

Λειτουργία

Χρησιμοποιείται για την εμφάνιση σταθερών τιμών καθώς και των τιμών των μεταβλητών που αναφέρονται στη λίστα.

Η εντολή **ΓΡΑΨΕ** έχει ως αποτέλεσμα την εμφάνιση τιμών στη μονάδα εξόδου. Συσκευή εξόδου μπορεί να είναι η οθόνη του υπολογιστή, ο εκτυπωτής, βοηθητική μνήμη ή γενικά οποιαδήποτε συσκευή εξόδου έχει οριστεί στο πρόγραμμα. Για τα παραδείγματα αυτού του κεφαλαίου θεωρούμε ότι η εμφάνιση γίνεται πάντοτε στην οθόνη. Η λίστα των στοιχείων μπορεί να περιέχει σταθερές τιμές και ονόματα μεταβλητών.

Κατά την εκτέλεση του προγράμματος η εντολή **ΓΡΑΨΕ** προκαλεί την εμφάνιση στην οθόνη των σταθερών τιμών. Όταν κάποιο όνομα μεταβλητής περιέχεται στη λίστα τότε αρχικά ανακτάται η τιμή της και στη συνέχεια η τιμή αυτή εμφανίζεται στην οθόνη.

Η χρήση της εντολής ΓΡΑΨΕ είναι κυρίως η εμφάνιση μηνυμάτων από τον υπολογιστή, καθώς και αποτελεσμάτων που περιέχονται στις μεταβλητές.

7.10 Δομή προγράμματος

Όπως κάθε εντολή ακολουθεί αυστηρούς συντακτικούς κανόνες, έτσι και ολόκληρο το πρόγραμμα έχει αυστηρούς κανόνες για τον τρόπο που δομείται. Η πρώτη εντολή κάθε προγράμματος είναι υποχρεωτικά η επικεφαλίδα του προγράμματος, η οποία είναι η λέξη **ΠΡΟΓΡΑΜΜΑ** ακολουθούμενη από το όνομα του προγράμματος. Το τελευταίο πρέπει να υπακούει στους κανόνες δημιουργίας ονομάτων της **ΓΛΩΣΣΑΣ**.

Στη συνέχεια ακολουθεί το τμήμα δήλωσης των σταθερών του προγράμματος, αν βέβαια το πρόγραμμα μας χρησιμοποιεί σταθερές.

Αμέσως μετά είναι το τμήμα δήλωσης μεταβλητών, όπου δηλώνονται υποχρεωτικά τα ονόματα όλων των μεταβλητών καθώς και ο τύπος τους.

Ακολουθεί το κύριο μέρος του προγράμματος, που περιλαμβάνει όλες τις εκτελέσιμες εντολές. Οι εντολές αυτές περιλαμβάνονται υποχρεωτικά ανάμεσα στις λέξεις **ΑΡΧΗ** και **ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ**.

Τέλος αν το πρόγραμμα χρησιμοποιεί διαδικασίες (βλ. κεφ. 10), αυτές γράφονται μετά το **ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ**.

Κάθε εντολή γράφεται σε ξεχωριστή γραμμή. Αν μία εντολή πρέπει να συνεχιστεί και στην επόμενη γραμμή, τότε ο πρώτος χαρακτήρας αυτής της γραμμής πρέπει να είναι ο χαρακτήρας &.

Αν ο πρώτος χαρακτήρας είναι το θαυμαστικό (!), σημαίνει ότι αυτή η γραμμή περιέχει σχόλια και όχι εκτελέσιμες εντολές.

Παράδειγμα

Το επόμενο πρόγραμμα υπολογίζει το συνολικό κόστος παραγγελιών υπολογιστών. Το πρόγραμμα διαβάζει από το πληκτρολόγιο την ποσότητα της παραγγελίας και την τιμή του ενός υπολογιστή, υπολογίζει και γράφει το συνολικό κόστος καθώς και το αντίστοιχο κόστος του ΦΠΑ. Ο συντελεστής ΦΠΑ είναι 18%.

```
ΠΡΟΓΡΑΜΜΑ Κόστος_Υπολογιστών
! Πρόγραμμα υπολογισμού κόστους παραγγελίας υπολογιστών
ΣΤΑΘΕΡΕΣ
ΦΠΑ=0.18
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: Ποσότητα, Τιμή_μονάδας, Κόστος
    ΠΡΑΓΜΑΤΙΚΕΣ: Αξία_ΦΠΑ, Συνολικό_κόστος
ΑΡΧΗ
! Εισαγωγή δεδομένων
ΓΡΑΨΕ 'Δώσε την ποσότητα της παραγγελίας'
ΔΙΑΒΑΣΕ Ποσότητα
ΓΡΑΨΕ 'Δώσε την τιμή του υπολογιστή'
ΔΙΑΒΑΣΕ Τιμή_μονάδας
! Υπολογισμοί
Κόστος <- Ποσότητα* Τιμή_μονάδας
Αξία_ΦΠΑ <- Κόστος*ΦΠΑ
Συνολικό_κόστος <- Κόστος+Αξία_ΦΠΑ
! Εμφάνιση αποτελεσμάτων
ΓΡΑΨΕ 'Το κόστος των',Ποσότητα,'υπολογ. είναι ',Κόστος
ΓΡΑΨΕ  ' Η αξία του ΦΠΑ  είναι', Αξία_ΦΠΑ
ΓΡΑΨΕ 'Το συνολικό κόστος είναι', Συνολικό_κόστος
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Στη συνέχεια παρουσιάζεται το πρόγραμμα αυτό σε γλώσσες προγραμματισμού Pascal και Basic.

Προγραμματιστικό περιβάλλον Pascal

```
PROGRAM computers;

CONST
    fpa=0.18;

VAR
    cost, value, quantity:LONGINT;
    total,cost_fpa:REAL;

BEGIN
    write('Δώσε την ποσότητα');
    readln(quantity);
    write('Δώσε την τιμή του υπολογιστή');
    readln(value);
    cost:=quantity*value;
```

```
    cost_fpa:=cost*fpa;
    total:=cost+cost_fpa;
    writeln ('Το κόστος των ', quantity, ' είναι:',
cost);
    writeln ('Η αξία του ΦΠΑ: ', cost_fpa:7:0);
    writeln ('Το συνολικό κόστος είναι: ', total:7:0)
END.
```

Προγραμματιστικό περιβάλλον Basic

```
' Κόστος υπολογιστών
fpa = .18
INPUT "Δώσε την ποσότητα : ", Quantity
INPUT "Δώσε την τιμή του υπολογιστή : ", Value
Cost = Quantity * Value
CostFpa = Cost * fpa
Total = Cost + CostFpa
PRINT "Το κόστος των"; Quantity; "υπολογιστών είναι :";
Cost
PRINT USING "Η αξία ΦΠΑ είναι : ######"; CostFpa
PRINT USING "Το συνολικό κόστος είναι : #######"; Total
END
```

Ανακεφαλαίωση

Σε αυτό το κεφάλαιο παρουσιάστηκαν τα βασικά χαρακτηριστικά της **ΓΛΩΣΣΑΣ**, το αλφάβητο της, οι τύποι δεδομένων που υποστηρίζει, οι κανόνες για τα ονόματα που χρησιμοποιούνται, οι αριθμητικές πράξεις, η εντολή εκχώρησης, οι εντολές εισόδου και εξόδου καθώς και η δομή που πρέπει να έχει κάθε πρόγραμμα.

Συγκεκριμένα:

- Οι τύποι δεδομένων που υποστηρίζονται είναι: Ακέραιοι, Πραγματικοί, Χαρακτήρες, Λογικοί.
- Οι μεταβλητές πρέπει να έχουν τον τύπο των δεδομένων που καταχωρούν.
- Κάθε μεταβλητή παίρνει τιμή με εντολή εκχώρησης ή με εντολή ΔΙΑΒΑΣΕ.
- Κάθε πρόγραμμα έχει τον τίτλο του, ακολουθεί το τμήμα δηλώσεων (σταθερών και μεταβλητών) και μετά ανάμεσα στις λέξεις ΑΡΧΗ και ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ βρίσκονται όλες οι εκτελέσιμες εντολές.

- Η επικοινωνία του προγράμματος με τον χρήστη γίνεται με τις εντολές εισόδου εξόδου ΓΡΑΨΕ και ΔΙΑΒΑΣΕ.

Λέξεις κλειδιά

Πρόγραμμα, Τύποι δεδομένων, Μεταβλητή, Σταθερά, Εντολή, Εκχώρηση τιμής, Είσοδος- έξοδος προγράμματος

Ερωτήσεις - Θέματα για συζήτηση

- Ποιους τύπους δεδομένων γνωρίζετε. Αναφέρατε δύο παραδείγματα για κάθε τύπο;
- Σε ποια θέση του προγράμματος αναγράφονται οι δηλώσεις των σταθερών;
- Ποια η διαφορά μεταβλητών και σταθερών;
- Ποια η σειρά εκτέλεσης των πράξεων;
- Ποιος ο σκοπός των εντολών εισόδου εξόδου;
- Ποια η διαφορά των εντολών ΔΙΑΒΑΣΕ και ΓΡΑΨΕ;
- Περιγράψτε τη δομή ενός προγράμματος;

Βιβλιογραφία

1. Θ. Αλεβίζος, Α. Καμπουρέλης, *Εισαγωγή με τη γλώσσα Pascal*, Αθήνα, 1984.
2. Γ. Βουτυράς, *Basic: Αλγόριθμοι και εφαρμογές*, Κλεψύδρα, Αθήνα, 1991.
3. Χρ. Κοίλιας, *Η QuickBasic και οι εφαρμογές της*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.
4. R. Shackelford, *Introduction to Computing and Algorithms*, Addison-Wesley, USA, 1998.
5. S. Leestma-L.Nyhoff, *Turbo Pascal, Programming and Solving*, McMillan, New York, 1990.
6. N. Wirth, *Systematic Programming: An introduction*, Prentice Hall, 1973.

Διευθύνσεις Διαδικτύου

- http://qbasic.com/

Περιέχει εκπαιδευτικό οδηγό, κώδικα πολλών ασκήσεων και γενικές πληροφορίες για την Qbasic.

- www.basicguru.com

Διεύθυνση που αναφέρεται αποκλειστικά στη Basic. Περιέχει πολλά έτοιμα παραδείγματα, πληροφορίες για εκδόσεις της γλώσσας, μεταφραστές για διάφορα λειτουργικά συστήματα.

ΚΕΦΑΛΑΙΟ 8ο

ΕΠΙΛΟΓΗ ΚΑΙ ΕΠΑΝΑΛΗΨΗ

ΠΕΡΙΕΧΟΜΕΝΑ

- Εντολές επιλογής
- Εντολές επανάληψης

Εισαγωγή

Στο προηγούμενο κεφάλαιο αναπτύξαμε προγράμματα, τα οποία ήταν πολύ απλά και οι εντολές των οποίων εκτελούνται η μία μετά την άλλη. Αυτή η σειριακή εκτέλεση των εντολών είναι κατάλληλη όμως μόνο για πολύ απλά προγράμματα, τα οποία εισάγουν δεδομένα, τα επεξεργάζονται και τυπώνουν το αποτέλεσμα, χωρίς να υπάρχει η δυνατότητα της επιλεκτικής εκτέλεσης τμημάτων του προγράμματος, σύμφωνα με την τιμή κάποιων δεδομένων ή την επανάληψη τμημάτων του προγράμματος. Όπως έχουμε αναφέρει οι τρεις βασικές δομές, είναι η δομή της ακολουθίας, της επιλογής και της επανάληψης. Οι δομές αυτές αποτελούν τη βάση του δομημένου προγραμματισμού και με τη χρήση αυτών μπορούν να υλοποιηθούν όλα τα προγράμματα υπολογιστών. Στο κεφάλαιο αυτό θα ασχοληθούμε με τις δύο αυτές βασικές δομές της επιλογής και της επανάληψης που θα μας επιτρέψουν την συγγραφή πληρέστερων και πιο πολύπλοκων προγραμμάτων.

Διδακτικοί στόχοι

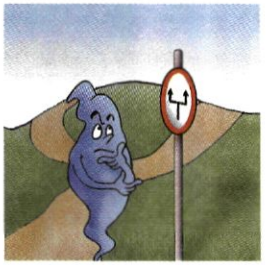
Να είναι σε θέση ο μαθητής:

- Να σχηματίζει λογικές εκφράσεις, απλές και σύνθετες.
- Να διατυπώνει τις μορφές της εντολής ελέγχου (επιλογής) ΑΝ.
- Να διακρίνει τις διαφορές των μορφών της εντολής ΑΝ.
- Να επιλέγει την καλύτερη μορφή της εντολής ΑΝ για το κάθε πρόγραμμα.
- Να διατυπώνει τις εντολές επανάληψης.
- Να επιλέγει την καλύτερη δομή επανάληψης και να χρησιμοποιεί την κατάλληλη εντολή.
- Να συντάσσει προγράμματα τα οποία χρησιμοποιούν και τις τρεις βασικές δομές: της ακολουθίας, της επιλογής και της επανάληψης.

Προερωτήσεις

- ✓ Γιατί χρησιμοποιούνται οι αλγοριθμικές δομές;
- ✓ Νομίζεις ότι όλες οι αλγοριθμικές δομές έχουν τις αντίστοιχες εντολές σε μία γλώσσα προγραμματισμού;
- ✓ Η δομή της επιλογής είναι σημαντική για την επίλυση προβλημάτων;
- ✓ Αρκεί μία εντολή για να εκφράσει την δομή της επανάληψης;
- ✓ Πώς μπορεί να ελέγχεται ο τερματισμός μίας επανάληψης;

8.1 Εντολές Επιλογής



Μία από τις βασικότερες δομές που εμφανίζονται σε ένα πρόγραμμα, είναι η επιλογή. Σχεδόν σε όλα τα προβλήματα περιλαμβάνονται κάποιοι έλεγχοι και ανάλογα με το αποτέλεσμα αυτών των ελέγχων επιλέγονται οι ενέργειες που θα ακολουθήσουν.

Ας θεωρήσουμε το πολύ απλό πρόβλημα της καταμέτρησης των θετικών και των αρνητικών αριθμών. Πρέπει λοιπόν να γράψουμε ένα πρόγραμμα, το οποίο εισάγει αριθμούς και μετράει πόσοι από αυτούς είναι θετικοί και πόσοι αρνητικοί. Για να αποφασίσουμε, αν ένας αριθμός είναι θετικός ή αρνητικός, πρέπει να τον συγκρίνουμε με το 0. Το αποτέλεσμα αυτής της σύγκρισης καθορίζει το είδος του αριθμού, αν είναι μεγαλύτερος από το 0, τότε ο αριθμός είναι θετικός, ενώ αντίθετα αν είναι μικρότερος από το 0, είναι αρνητικός.

Λογική Έκφραση

Για τη σύνταξη μιας λογικής έκφρασης ή συνθήκης χρησιμοποιούνται σταθερές, μεταβλητές, αριθμητικές παραστάσεις, συγκριτικοί και λογικοί τελεστές, καθώς και παρενθέσεις. Στις λογικές εκφράσεις γίνεται σύγκριση της τιμής μίας έκφρασης, που βρίσκεται αριστερά από το συγκριτικό τελεστή με την τιμή μιας άλλης έκφρασης που βρίσκεται δεξιά. Το αποτέλεσμα είναι μία λογική τιμή **ΑΛΗΘΗΣ** ή **ΨΕΥΔΗΣ**.

Οι χρησιμοποιούμενοι συγκριτικοί τελεστές παρουσιάζονται στον επόμενο πίνακα.

Συγκριτικοί τελεστές		
Τελεστής	Ελεγχόμενη σχέση	Παράδειγμα
=	Ισότητα	Αριθμός=0
<>	Ανισότητα	Ονομα1 <> 'Κώστας'
>	Μεγαλύτερο από	Τιμή>10000
>=	Μεγαλύτερο ή ίσο	Χ+Υ >= (Α+Β)/Γ
<	Μικρότερο από	Β^2-4*Α*Γ < 0
<=	Μικρότερο ή ίσο	Βάρος <= 500

Οι συγκρίσεις γίνονται σε δεδομένα αριθμητικά, αλφαριθμητικά και λογικά.

Η σύγκριση μεταξύ δύο αριθμών γίνεται με προφανή τρόπο. Στην περίπτωση των πραγματικών αριθμών θεωρούμε ότι οι αριθμοί μπορούν να έχουν άπειρο αριθμό ψηφίων.

Η σύγκριση ατομικών χαρακτήρων στηρίζεται στην αλφαβητική σειρά, για παράδειγμα το 'α' θεωρείται μικρότερο από το 'β'.

Η σύγκριση αλφαριθμητικών δεδομένων βασίζεται στη σύγκριση χαρακτήρα προς χαρακτήρα σε κάθε θέση μέχρις ότου βρεθεί κάποια διαφορά, για παράδειγμα η λέξη 'κακός' θεωρείται μικρότερη από τη λέξη 'καλός' αφού το γράμμα κ προηγείται του γράμματος λ.

Όταν αριθμητικοί και συγκριτικοί τελεστές συνδυάζονται σε μια έκφραση, οι αριθμητικές πράξεις εκτελούνται πρώτες. Ακόμη, οι λογικοί τελεστές έχουν χαμηλότερη ιεραρχία από τους συγκριτικούς.

Η σύγκριση λογικών έχει έννοια μόνο στην περίπτωση του ίσου (=) και του διάφορου (<>), αφού οι τιμές που μπορούν να έχουν είναι ΑΛΗΘΗΣ και ΨΕΥΔΗΣ.

Σύνθετες Εκφράσεις

Σε πολλά προβλήματα οι επιλογές δεν αρκεί να γίνονται με απλές λογικές παραστάσεις όπως αυτές οι οποίες αναφέρθηκαν, αλλά χρειάζεται να συνδυαστούν μία ή περισσότερες λογικές παραστάσεις. Αυτό επιτυγχάνεται με τη χρήση των τριών βασικών λογικών τελεστών ΟΧΙ, ΚΑΙ, 'Η.

Παραδείγματα

0<X<5 X>0 ΚΑΙ X<5

X=1 ή 2 ή 3 X=1 'Η X=2 'Η X=3

Η ιεραρχία των λογικών τελεστών είναι μικρότερη των αριθμητικών.

8.1.1 Εντολή ΑΝ

Η δομή επιλογής υλοποιείται στη **ΓΛΩΣΣΑ** με την εντολή **ΑΝ**. Η εντολή **ΑΝ** εμφανίζεται με τρεις διαφορετικές μορφές. Την απλή εντολή **ΑΝ...ΤΟΤΕ**, την εντολή **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ** και τέλος την εντολή **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ_ΑΝ.** Κάθε εντολή **ΑΝ** πρέπει να κλείνει με **ΤΕΛΟΣ_ΑΝ.**

Στην απλούστερη μορφή της η εντολή **ΑΝ** ελέγχει τη συνθήκη και αν αυτή ισχύει (είναι αληθής), τότε εκτελούνται οι εντολές που περιλαμβάνονται μεταξύ των λέξεων **ΤΟΤΕ** και **ΤΕΛΟΣ_ΑΝ**.

Αν για παράδειγμα θέλουμε να υπολογίσουμε την τετραγωνική ρίζα των αριθμών που διαβάζουμε από το πληκτρολόγιο, τότε το αντίστοιχο τμήμα προγράμματος είναι.

```
ΔΙΑΒΑΣΕ α
ΑΝ  α >=0 ΤΟΤΕ
    Ρίζα <- Τ_Ρ(α)
ΤΕΛΟΣ_ΑΝ
```

Η γενική μορφή της εντολής ΑΝ είναι η εξής:

Σύνταξη

```
ΑΝ συνθήκη  ΤΟΤΕ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΤΕΛΟΣ_ΑΝ
```

Παράδειγμα

```
ΑΝ αριθμός > 0  ΤΟΤΕ
ΓΡΑΨΕ 'Ο αριθμός είναι θετικός'
    Πλήθος_θετικών <- Πλήθος_θετικών+1
ΤΕΛΟΣ_ΑΝ
```

Λειτουργία

Αν η συνθήκη ισχύει, τότε εκτελούνται οι εντολές που βρίσκονται μεταξύ των λέξεων **ΤΟΤΕ** και **ΤΕΛΟΣ_ΑΝ**, σε αντίθετη περίπτωση οι εντολές αυτές αγνοούνται. Η εκτέλεση του προγράμματος συνεχίζεται με την εντολή που ακολουθεί τη δήλωση **ΤΕΛΟΣ_ΑΝ**

Συχνά η εντολή **ΑΝ** εκτός από το τμήμα των εντολών, που εκτελούνται όταν η λογική έκφραση είναι Αληθής, περιέχει και το τμήμα των εντολών που εκτελούνται, αν δεν ισχύει η συνθήκη (είναι Ψευδής).

Η μορφή αυτής της εντολής ονομάζεται **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ**.

Στο παράδειγμα του υπολογισμού της τετραγωνικής ρίζας έχουμε

```
ΔΙΑΒΑΣΕ α
ΑΝ α >=0 ΤΟΤΕ
    Ρίζα <- Τ_Ρ(α)
ΑΛΛΙΩΣ
    ΓΡΑΨΕ ' Η τετρ. ρίζα αρνητικού αριθμού δεν ορίζεται'
ΤΕΛΟΣ_ΑΝ
```

Η γενική μορφή της εντολής **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ** έχει ως εξής:

Σύνταξη

```
ΑΝ συνθήκη   ΤΟΤΕ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΑΛΛΙΩΣ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΤΕΛΟΣ_ΑΝ
```

Παράδειγμα

```
ΑΝ αριθμός > 0 ΤΟΤΕ
    ΓΡΑΨΕ 'Ο αριθμός είναι θετικός'
    Πλήθος_θετικών <- Πλήθος_θετικών+1
ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Ο αριθμός είναι αρνητικός ή 0'
    Πλήθος_μη_θετικών <- Πλήθος_μη_θετικών +1
ΤΕΛΟΣ_ΑΝ
```

Λειτουργία

Αν η συνθήκη ισχύει, τότε εκτελούνται οι εντολές που βρίσκονται μεταξύ των λέξεων **ΤΟΤΕ** και **ΑΛΛΙΩΣ**, διαφορετικά εκτελούνται οι εντολές μεταξύ **ΑΛΛΙΩΣ** και **ΤΕΛΟΣ_ΑΝ**. Η εκτέλεση του προγράμματος συνεχίζεται με την εντολή που ακολουθεί τη δήλωση **ΤΕΛΟΣ_ΑΝ**

Η γενική μορφή της εντολής **ΑΝ** καλύπτει την επιλογή μιας από δύο εναλλακτικές περιπτώσεις.

Όταν οι εναλλακτικές περιπτώσεις είναι περισσότερες από τις δύο, τότε μπορούν να χρησιμοποιηθούν πολλές εντολές **ΑΝ** η μία μέσα στην άλλη, οι εμφωλευμένες εντολές **ΑΝ**, όπως ονομάζονται.

Εμφωλευμένα ΑΝ ονομάζονται δύο ή περισσότερες εντολές της μορφής **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ** που περιέχονται η μία μέσα στην άλλη.

Για παράδειγμα οι παρακάτω εντολές προγράμματος

```
''''''''
ΔΙΑΒΑΣΕ Βάρος, Ύψος
ΑΝ Βάρος < 80 ΤΟΤΕ
    ΑΝ Ύψος < 1.70 ΤΟΤΕ
        ΓΡΑΨΕ  'Ελαφρύς, κοντός'
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
''''''
```

Η χρήση εμφωλευμένων εντολών **ΑΝ** οδηγεί συνήθως σε πολύπλοκες δομές που αυξάνουν την πιθανότητα του λάθους καθώς και τη δυσκολία κατανόησης του προγράμματος.

Πολύ συχνά οι εντολές που έχουν γραφεί με εμφωλευμένα **ΑΝ,** μπορούν να γραφούν πιο απλά χρησιμοποιώντας σύνθετες εκφράσεις ή την εντολή επιλογής **ΑΝ ... ΤΟΤΕ ... ΑΛΛΙΩΣ_ΑΝ**, που θα παρουσιαστεί στη συνέχεια.

Το προηγούμενο τμήμα προγράμματος μπορεί να γραφεί ως εξής

```
''''''''
ΔΙΑΒΑΣΕ Βάρος, Ύψος
ΑΝ Βάρος < 80  ΚΑΙ  Ύψος < 1.70 ΤΟΤΕ
    ΓΡΑΨΕ  'Ελαφρύς, κοντός'
ΤΕΛΟΣ_ΑΝ
''''''
```

Μία άλλη μορφή επιλογής είναι η εντολή **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ_ΑΝ**

Σύνταξη

```
ΑΝ συνθήκη-1  ΤΟΤΕ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΑΛΛΙΩΣ_ΑΝ συνθήκη-2  ΤΟΤΕ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
...
ΑΛΛΙΩΣ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΤΕΛΟΣ_ΑΝ
```

Παράδειγμα

```
ΑΝ αριθμός > 0 ΤΟΤΕ
    ΓΡΑΨΕ 'Ο αριθμός είναι θετικός'
    Πλήθος-θετικών <- Πλήθος_θετικών+1
ΑΛΛΙΩΣ_ΑΝ αριθμός <0 ΤΟΤΕ
    ΓΡΑΨΕ 'Ο αριθμός είναι αρνητικός '
    Πλήθος_αρνητικών <- Πλήθος_ αρνητικών +1
ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Ο αριθμός είναι 0'
    Πλήθος_0 <- Πλήθος_0 +1
ΤΕΛΟΣ_ΑΝ
```

Λειτουργία

Εκτελούνται οι εντολές που βρίσκονται στο αντίστοιχο τμήμα, όταν η συνθήκη είναι αληθής.

Η εκτέλεση του προγράμματος συνεχίζεται με την εντολή που ακολουθεί τη δήλωση **ΤΕΛΟΣ_ΑΝ**

Παράδειγμα 1

Στο πρόγραμμα του προηγούμενου κεφαλαίου (πωλήσεις υπολογιστών) υποθέτουμε ότι η τιμή των υπολογιστών εξαρτάται από την ποσότητα παραγγελίας. Συγκεκριμένα ισχύουν οι παρακάτω τιμές αγοράς υπολογιστών.

ΠΟΣΟΤΗΤΑ	ΤΙΜΗ ΜΟΝΑΔΑΣ
1-50	580
51-100	520
101-200	470
πάνω από 200	440

Ο υπολογισμός με χρήση εμφωλευμένων εντολών ΑΝ είναι:

```
ΑΝ Ποσότητα<=50 ΤΟΤΕ
    Κόστος <- Ποσότητα*580
ΑΛΛΙΩΣ
    ΑΝ Ποσότητα<=100 ΤΟΤΕ
        Κόστος <- Ποσότητα*520
    ΑΛΛΙΩΣ
        ΑΝ Ποσότητα<=200 ΤΟΤΕ
            Κόστος <- Ποσότητα*470
        ΑΛΛΙΩΣ
            Κόστος <- Ποσότητα*440
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΑΝ
```

Το ίδιο πρόγραμμα με τη χρήση της εντολής **ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ_ΑΝ** έχει ως εξής:

```
ΑΝ Ποσότητα<=50 ΤΟΤΕ
    Κόστος <- Ποσότητα*580
ΑΛΛΙΩΣ_ΑΝ Ποσότητα<=100 ΤΟΤΕ
    Κόστος <- Ποσότητα*520
ΑΛΛΙΩΣ_ΑΝ Ποσότητα<=200 ΤΟΤΕ
    Κόστος <- Ποσότητα*470
ΑΛΛΙΩΣ
    Κόστος <- Ποσότητα*440
ΤΕΛΟΣ_ΑΝ
```

Να αποφεύγεται, αν είναι δυνατόν, η χρήση των εμφωλευμένων ΑΝ, και στη θέση τους να χρησιμοποιούνται απλούστερες δομές που διευκολύνουν την ανάγνωση και την κατανόηση του προγράμματος.

Ένα συχνό λάθος που παρατηρείται στα προγράμματα είναι ο έλεγχος περιττών συνθηκών. Οι επιπλέον έλεγχοι αυξάνουν την πολυπλοκότητα του προγράμματος.

Στο προηγούμενο παράδειγμα για το οποίο θεωρούμε ότι η ποσότητα είναι θετικός αριθμός, ένα παράδειγμα περιττών ελέγχων είναι το ακόλουθο:

Στην πρώτη εντολή ΑΛΛΙΩΣ_ΑΝ ο έλεγχος της συνθήκης Ποσότητα>50 είναι περιττός

```
ΑΝ Ποσότητα<=50 ΤΟΤΕ
    Κόστος <- Ποσότητα* 580
ΑΛΛΙΩΣ_ΑΝ Ποσότητα>50 ΚΑΙ Ποσότητα <=100 ΤΟΤΕ
    Κόστος <- Ποσότητα* 520
ΑΛΛΙΩΣ_ΑΝ Ποσότητα>100 ΚΑΙ Ποσότητα <=200 ΤΟΤΕ
    Κόστος <- Ποσότητα* 470
ΑΛΛΙΩΣ
    Κόστος <- Ποσότητα* 440
ΤΕΛΟΣ_ΑΝ
```

8.1.2 Εντολή ΕΠΙΛΕΞΕ

Αν οι εναλλακτικές περιπτώσεις επιλογής είναι πολλές, μπορεί να χρησιμοποιηθεί η εντολή ΕΠΙΛΕΞΕ, η γενική μορφή της οποίας είναι:

Σύνταξη

```
ΕΠΙΛΕΞΕ έκφραση
    ΠΕΡΙΠΤΩΣΗ λίστα_τιμών_1
        εντολές_1
    ΠΕΡΙΠΤΩΣΗ  λίστα_τιμών_2
        εντολές_2
    ......
    ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
        εντολές_αλλιώς
ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
```

Παράδειγμα

```
ΔΙΑΒΑΣΕ αριθμός
ΕΠΙΛΕΞΕ αριθμός
    ΠΕΡΙΠΤΩΣΗ 0
        ΓΡΑΨΕ 'Μηδέν'
```

```
        ΠΕΡΙΠΤΩΣΗ 1,3,5,7,9
            ΓΡΑΨΕ 'Μονός αριθμός'
        ΠΕΡΙΠΤΩΣΗ 2,4,6,8
            ΓΡΑΨΕ 'Ζυγός '
        ΠΕΡΙΠΤΩΣΗ ΑΛΛΙΩΣ
            ΓΡΑΨΕ 'Αριθμός < 0 ή >9 ή όχι ακέραιος'
    ΤΕΛΟΣ_ΕΠΙΛΟΓΩΝ
```

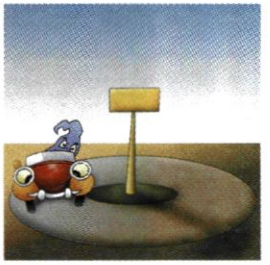
Λειτουργία

Υπολογίζεται η τιμή της έκφρασης και εκτελούνται οι εντολές που ανήκουν στην αντίστοιχη περίπτωση τιμών. Αν η τιμή της έκφρασης δεν αντιστοιχεί σε καμία περίπτωση, τότε εκτελούνται οι εντολές αλλιώς.

Στην εντολή αυτή οι λίστες τιμών που συνοδεύουν κάθε περίπτωση μπορούν να περιλαμβάνουν μία ή περισσότερες τιμές ή περιοχή τιμών από-έως.

Η χρήση της εντολής **ΕΠΙΛΕΞΕ** λόγω της συμπαγούς δομής της προσφέρει σημαντικά πλεονεκτήματα στον προγραμματισμό.

8.2 Εντολές επανάληψης



Η τρίτη βασική δομή είναι η δομή επανάληψης, ο βρόχος, η οποία επιτρέπει την εκτέλεση εντολών περισσότερες από μία φορά. Οι επαναλήψεις ελέγχονται πάντοτε από κάποια συνθήκη, η οποία καθορίζει την έξοδο από το βρόχο.

Η **ΓΛΩΣΣΑ** υποστηρίζει τρεις εντολές επανάληψης, την εντολή **ΟΣΟ** όπου η επανάληψη ελέγχεται από μία λογική έκφραση στην αρχή και εκτελείται συνεχώς όσο η συνθήκη είναι Αληθής, την εντολή **ΜΕΧΡΙΣ_ΟΤΟΥ** όπου η συνθήκη βρίσκεται στο τέλος του βρόχου και εκτελείται συνεχώς μέχρις ότου η συνθήκη αυτή γίνει Αληθής και τέλος την εντολή **ΓΙΑ**, με την οποία ο βρόχος επαναλαμβάνεται για προκαθορισμένο αριθμό φορών.

8.2.1 Εντολή ΟΣΟ...ΕΠΑΝΑΛΑΒΕ

Η γενικότερη δομή επανάληψης υλοποιείται στη **ΓΛΩΣΣΑ** με την εντολή **ΟΣΟ... ΕΠΑΝΑΛΑΒΕ**. Σε αυτή, η συνθήκη που ελέγχει την επανάληψη βρίσκεται στην αρχή της επανάληψης και ο βρόχος επαναλαμβάνεται συνεχώς, όσο η συνθήκη αυτή ισχύει. Με τη δομή αυτή μπορούν να εκφραστούν

όλες οι επαναλήψεις και γι αυτό η εντολή **ΟΣΟ... ΕΠΑΝΑΛΑΒΕ** είναι η σημαντικότερη από όλες τις εντολές επανάληψης. Χαρακτηριστικό της επανάληψης αυτής είναι ότι ο αριθμός των επαναλήψεων δεν είναι γνωστός, ούτε μπορεί να υπολογιστεί πριν από την εκτέλεση του προγράμματος.

Σύνταξη

```
ΟΣΟ συνθήκη ΕΠΑΝΑΛΑΒΕ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Παράδειγμα

```
Άθροισμα<-0
ΟΣΟ Άθροισμα<1000 ΕΠΑΝΑΛΑΒΕ
    ΔΙΑΒΑΣΕ Α
    Άθροισμα<- Άθροισμα+Α
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Λειτουργία

Ελέγχεται η συνθήκη και αν είναι Αληθής, εκτελούνται οι εντολές που βρίσκονται ανάμεσα στις **ΟΣΟ_ΕΠΑΝΑΛΑΒΕ** και **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ.** Στη συνέχεια ελέγχεται πάλι η συνθήκη και αν ισχύει, εκτελούνται πάλι οι ίδιες εντολές. 'Οταν η λογική έκφραση γίνει Ψευδής, τότε σταματάει η επανάληψη και εκτελείται η εντολή μετά το **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ**.

Εφόσον μετά από κάθε επανάληψη ελέγχεται εκ νέου η συνθήκη, πρέπει υποχρεωτικά μέσα στο βρόχο να υπάρχει μία εντολή, η οποία να μεταβάλει την τιμή της μεταβλητής που ελέγχεται με τη συνθήκη. Σε αντίθετη περίπτωση η επανάληψη δε θα τερματίζεται και θα εκτελείται συνεχώς.

Παράδειγμα 2

Να γραφεί πρόγραμμα το οποίο διαβάζει από το πληκτρολόγιο μία σειρά μετρήσεων, ακεραίων μη μηδενικών αριθμών, υπολογίζει και τυπώνει το άθροισμα τους καθώς και το μέσο τους όρο. Ως τέλος της διαδικασίας εισαγωγής στοιχείων χρησιμοποιείται η τιμή 0.

```
ΠΡΟΓΡΑΜΜΑ Άθροισμα
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ:Χ, Άθροισμα, Πλήθος
    ΠΡΑΓΜΑΤΙΚΕΣ:ΜΟ
ΑΡΧΗ
Πλήθος <- 0
Άθροισμα <- 0
ΓΡΑΨΕ 'Δώσε Αριθμό'
ΔΙΑΒΑΣΕ Χ
ΟΣΟ Χ<>0 ΕΠΑΝΑΛΑΒΕ
    Άθροισμα <- Άθροισμα+Χ
    Πλήθος <- Πλήθος+1
    ΓΡΑΨΕ 'Δώσε Αριθμό'
    ΔΙΑΒΑΣΕ Χ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΑΝ Πλήθος >0 ΤΟΤΕ
    ΜΟ <- Άθροισμα/Πλήθος
    ΓΡΑΨΕ 'Το Άθροισμα είναι   : ', Άθροισμα
    ΓΡΑΨΕ  'Ο Μέσος όρος είναι : ', ΜΟ
ΑΛΛΙΩΣ
    ΓΡΑΨΕ 'Δεν δόθηκαν στοιχεία'
ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Παρατηρήσεις

Η χρήση τιμών για τον τερματισμό μίας επαναληπτικής διαδικασίας, όπως στο παράδειγμα η αυθαίρετη επιλογή του 0, είναι συνήθης στον προγραμματισμό.

Η τιμή αυτή ορίζεται από τον προγραμματιστή και αποτελεί μια σύμβαση για το τέλος του προγράμματος. Η τιμή αυτή είναι τέτοια, ώστε να μην είναι λογικά σωστή για το πρόβλημα, για παράδειγμα η τιμή 0 αποκλείεται από τις μετρήσεις σύμφωνα με την εκφώνηση του παραδείγματος. Η τιμή αυτή συχνά αποκαλείται "τιμή φρουρός".

8.2.2 Εντολή ΜΕΧΡΙΣ_ΟΤΟΥ

Η δεύτερη εντολή επανάληψης που χρησιμοποιεί η **ΓΛΩΣΣΑ** είναι η εντολή **ΜΕΧΡΙΣ_ΟΤΟΥ**. Σε αυτή οι εντολές του βρόχου εκτελούνται μέχρις ότου ικανοποιηθεί κάποια συνθήκη η οποία ελέγχεται στο τέλος της επανάληψης.

Σύνταξη

```
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΜΕΧΡΙΣ_ΟΤΟΥ λογική-έκφραση
```

Παράδειγμα

```
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΔΙΑΒΑΣΕ Α
    Άθροισμα <- Άθροισμα + Α
ΜΕΧΡΙΣ_ΟΤΟΥ Άθροισμα >= 1000
```

Λειτουργία

Εκτελούνται οι εντολές μεταξύ των **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ** και **ΜΕΧΡΙΣ_ΟΤΟΥ. Στ**η συνέχεια ελέγχεται η λογική έκφραση και αν δεν ισχύει (είναι ψευδής), τότε οι εντολές που βρίσκονται ανάμεσα στις **ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ** και **ΜΕΧΡΙΣ_ΟΤΟΥ,** εκτελούνται πάλι. Ελέγχεται ξανά η λογική έκφραση και αν δεν ισχύει, επαναλαμβάνεται η εκτέλεση των ίδιων εντολών.

Όταν η λογική έκφραση γίνει Αληθής τότε σταματάει η επανάληψη και εκτελείται η εντολή μετά από την **ΜΕΧΡΙΣ_ΟΤΟΥ**.

Πολύ συχνά η ίδια επαναληπτική διαδικασία μπορεί να γραφεί εξίσου σωστά χρησιμοποιώντας είτε τη δομή **ΟΣΟ...ΕΠΑΝΑΛΑΒΕ** είτε τη δομή **ΜΕΧΡΙΣ_ΟΤΟΥ** και είναι προσωπική επιλογή του προγραμματιστή ποια από τις δυο θα χρησιμοποιήσει. Υπάρχουν όμως περιπτώσεις όπου η χρήση της εντολής **ΜΕΧΡΙΣ_ΟΤΟΥ** οδηγεί σε απλούστερα και πιο ευκολονόητα προγράμματα. Γενικά σε περιπτώσεις όπου η επανάληψη θα συμβεί υποχρεωτικά μία φορά, είναι προτιμότερη η χρήση της **ΜΕΧΡΙΣ_ΟΤΟΥ**.

Χαρακτηριστική περίπτωση όπου προτιμάται η εντολή **ΜΕΧΡΙΣ_ΟΤΟΥ** είναι στον έλεγχο αποδεκτών τιμών καθώς και στην επιλογή από προκαθορισμένες απαντήσεις ή μενού.

Παράδειγμα 3

Στο προηγούμενο παράδειγμα ας υποθέσουμε ότι οι μετρήσεις είναι υποχρεωτικά θετικοί αριθμοί και ότι μετά την εισαγωγή κάθε αριθμού υπάρχει η ερώτηση, αν θα εισάγουμε άλλο. Η διαδικασία θα τελειώσει, όταν η απάντηση θα είναι Όχι (ο ή Ο).

```
ΠΡΟΓΡΑΜΜΑ Αθροισμα2
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ:Χ, Άθροισμα, Πλήθος
    ΠΡΑΓΜΑΤΙΚΕΣ:ΜΟ
    ΧΑΡΑΚΤΗΡΕΣ:Επιλογή
ΑΡΧΗ
Πλήθος <- 0
Άθροισμα <-0
ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
    ΑΡΧΗ_ΕΠΑΝΑΛΗΨΗΣ
        ! Έλεγχος δεδομένων
        ΓΡΑΨΕ 'Δώσε Αριθμό'
        ΔΙΑΒΑΣΕ Χ
        ΑΝ Χ=<0 ΤΟΤΕ
            ΓΡΑΨΕ 'Λάθος Αριθμός, Παρακαλώ δώστε ξανά...'
        ΤΕΛΟΣ_ΑΝ
        ! Αν το Χ δεν είναι θετικό εισάγουμε νέο αριθμό
    ΜΕΧΡΙΣ_ΟΤΟΥ Χ>0
    Άθροισμα <- Άθροισμα+Χ
    Πλήθος <- Πλήθος+1
    ΓΡΑΨΕ 'Νέα μέτρηση ;'
    ΔΙΑΒΑΣΕ Επιλογή
! Αν η επιλογή είναι Ο ή ο τότε σταματάει η επανάληψη
ΜΕΧΡΙΣ_ΟΤΟΥ Επιλογή='Ο' Ή Επιλογή='ο'
ΜΟ <- Άθροισμα/Πλήθος
ΓΡΑΨΕ 'Άθροισμα =', Άθροισμα
ΓΡΑΨΕ 'Μέσος όρος =', ΜΟ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Η εντολή επανάληψης ΜΕΧΡΙΣ_ΟΤΟΥ εκτελείται υποχρεωτικά τουλάχιστον μία φορά

Προγραμματιστικό περιβάλλον Pascal

```
PROGRAM athroisma2;
VAR
    x,sum,count:INTEGER;
    avg:REAL;
    choice:CHAR;
```

```
BEGIN
    count:=0;sum:=0;
    REPEAT
        REPEAT
            write('Δώσε αριθμό:');
            readln (x);
            IF x<=0 THEN
                writeln ('Λάθος αριθμός, Δώσε ξανά..');
        UNTIL x>0;
        sum:=sum+x;
        count:=count+1;
        write('Νέα μέτρηση;');
        readln(choice);
    UNTIL (choice='o') OR (choice='O');
    avg:=sum/count;
    writeln('Άθροισμα: ', sum:5);
    writeln('Μέσος όρος: ',avg:6:2);
END.
```

8.2.3 Εντολή ΓΙΑ...ΑΠΟ...ΜΕΧΡΙ

Πολύ συχνά ο αριθμός των επαναλήψεων που πρέπει να εκτελεστούν, είναι γνωστός από την αρχή. Αν και αυτού του είδους οι επαναλήψεις μπορούν να αντιμετωπιστούν με τη χρήση των προηγούμενων εντολών επανάληψης, η **ΓΛΩΣΣΑ** διαθέτει και την εντολή **ΓΙΑ**. Η εντολή αυτή χειρίζεται μια μεταβλητή, στην οποία αρχικά εκχωρείται η αρχική τιμή. Η τιμή της μεταβλητής συγκρίνεται με την τελική τιμή και εφόσον είναι μικρότερη από αυτή, τότε εκτελούνται οι εντολές που βρίσκονται στο βρόχο (ανάμεσα στις εντολές **ΓΙΑ** και **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ**). Στη συνέχεια η μεταβλητή ελέγχου αυξάνεται κατά την τιμή που ορίζει το **ΒΗΜΑ**. Αν η νέα τιμή είναι μικρότερη της τελικής, τότε ο βρόχος εκτελείται ξανά. Η διαδικασία αυτή επαναλαμβάνεται συνεχώς, έως ότου η τιμή ελέγχου γίνει μεγαλύτερη της τελικής τιμής, οπότε τερματίζεται η επανάληψη και το πρόγραμμα συνεχίζει με την εντολή που ακολουθεί το **ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ**.

Η εντολή ΓΙΑ...ΑΠΟ...ΜΕΧΡΙ χρησιμοποιείται στην περίπτωση που πρέπει να επαναληφθεί η εκτέλεση κάποιων εντολών για προκαθορισμένο αριθμό επαναλήψεων.

Ας σημειωθεί ότι, αν η τιμή του βήματος είναι 1, τότε μπορεί να παραληφθεί.

Σύνταξη

```
ΓΙΑ μεταβλητή ΑΠΟ τιμή1 ΜΕΧΡΙ τιμή2 ΜΕ ΒΗΜΑ τιμή3
    εντολή-1
    εντολή-2
    ...
    εντολή-ν
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

Παράδειγμα

```
ΓΙΑ Αριθμό ΑΠΟ 1 ΜΕΧΡΙ 100 ΜΕ ΒΗΜΑ 2
    Άθροισμα <-  Άθροισμα+Αριθμό
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

ΛΕΙΤΟΥΡΓΙΑ

Οι εντολές του βρόχου εκτελούνται για όλες τις τιμές της μεταβλητής από την αρχική τιμή μέχρι την τελική τιμή, αυξανόμενες με την τιμή του βήματος. Αν το βήμα είναι ίσο με 1, τότε παραλείπεται.

Παράδειγμα 4

Το παρακάτω πρόγραμμα υπολογίζει το άθροισμα των περιττών αριθμών που είναι μικρότεροι από το 100.

```
ΠΡΟΓΡΑΜΜΑ Περιττοί
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ:Άθροισμα, Αριθμός
ΑΡΧΗ
    Άθροισμα <- 0
    ΓΙΑ Αριθμός ΑΠΟ 1 ΜΕΧΡΙ 100 ΜΕ ΒΗΜΑ 2
        Άθροισμα <-  Άθροισμα + Αριθμός
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Άθροισμα περιττών αριθμών είναι: ', Άθροισμα
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Κάθε επανάληψη που εκτελείται με μία εντολή ΓΙΑ..ΑΠΟ..ΜΕΧΡΙ, μπορεί να υλοποιηθεί και με τη χρήση των βασικών εντολών επανάληψης ΟΣΟ..ΕΠΑΝΑΛΑΒΕ και ΜΕΧΡΙΣ..ΟΤΟΥ.

Πολύ συχνά για την επίλυση των προβλημάτων απαιτείται η χρήση εμφωλευμένων βρόχων. Σε αυτή την περίπτωση ο ένας βρόχος βρίσκεται μέσα στον άλλο.

Στη χρήση των εμφωλευμένων βρόχων ισχύουν συγκεκριμένοι κανόνες που πρέπει να ακολουθούνται αυστηρά για την σωστή λειτουργία των προγραμμάτων.

Συγκεκριμένα:

- Ο εσωτερικός βρόχος πρέπει να βρίσκεται ολόκληρος μέσα στον εξωτερικό. Ο βρόχος που ξεκινάει τελευταίος, πρέπει να ολοκληρώνεται πρώτος.
- Η είσοδος σε κάθε βρόχο υποχρεωτικά γίνεται από την αρχή του.
- Δεν μπορεί να χρησιμοποιηθεί η ίδια μεταβλητή ως μετρητής δύο ή περισσοτέρων βρόχων που ο ένας βρίσκεται στο εσωτερικό του άλλου.

Παράδειγμα 5

Να γραφεί πρόγραμμα το οποίο να εκτυπώνει τη προπαίδεια του πολλαπλασιασμού.

```
ΠΡΟΓΡΑΜΜΑ Προπαίδεια
!Πρόγραμμα εκτύπωσης της προπαίδειας των αριθμών 1 έως 10
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ:Α,Β,Γ
!Α:Πολλαπλασιαστέος
!Β:Πολαπλασιαστής
!Γ:Γινόμενο
ΑΡΧΗ
    ΓΙΑ  Α ΑΠΟ 1 ΜΕΧΡΙ 10
        ΓΙΑ Β ΑΠΌ 1 ΜΕΧΡΙ 10
            Γ <- Α*Β
            ΓΡΑΨΕ Α,'Χ',Β,'=',Γ
        ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
        ΓΡΑΨΕ !Εισαγωγή κενής γραμμής στην εκτύπωση
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Προγραμματιστικό περιβάλλον Basic

```
' Προπαίδεια
DEFINT A-Z
FOR a = 1 TO 10
  FOR b = 1 TO 10
    c = a * b
    PRINT a; "x"; b; "="; c
  NEXT b
```

```
  PRINT
NEXT a
END
```

Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάστηκαν οι εντολές που χρησιμοποιεί η ΓΛΩΣΣΑ για να υλοποιήσει τις βασικές δομές της επιλογής και της επανάληψης.

Αρχικά παρουσιάζονται οι λογικές εκφράσεις καθώς και ο τρόπος που διατυπώνονται σύνθετες λογικές εκφράσεις με τη χρήση των λογικών τελεστών ΟΧΙ, Ή, ΚΑΙ. Η εντολή ΑΝ...ΤΟΤΕ υλοποιεί τη δομή της επιλογής.

Η εντολή αυτή εμφανίζεται με πιο σύνθετες μορφές: την εντολή ΑΝ...ΤΟΤΕ...ΑΛΛΙΩΣ καθώς και την εντολή ΑΝ...ΤΟΤΕ... ΑΛΛΙΩΣ_ΑΝ. Μία άλλη εντολή επιλογής που υπάρχει είναι η εντολή ΕΠΙΛΕΞΕ.

Οι εντολές επανάληψης είναι τρεις. Η εντολή ΟΣΟ...ΕΠΑΝΑΛΑΒΕ, η εντολή ΜΕΧΡΙΣ_ΟΤΟΥ και τέλος η εντολή ΓΙΑ. Η εντολή ΓΙΑ χρησιμοποιείται για καθορισμένο αριθμό επαναλήψεων, ενώ ο αριθμός επαναλήψεων των άλλων δύο δεν είναι γνωστός εκ των προτέρων και εξαρτάται από τις συνθήκες που τις ελέγχουν. Η εντολή ΟΣΟ...ΕΠΑΝΑΛΑΒΕ ελέγχει τη συνθήκη στην αρχή της επανάληψης, ενώ η εντολή ΜΕΧΡΙΣ_ΟΤΟΥ κάνει τον έλεγχο στο τέλος της επανάληψης.

Λέξεις κλειδιά

Λογική έκφραση, Επιλογή, Επανάληψη, Βρόχος

Ερωτήσεις - Θέματα για συζήτηση

1. Ποιες είναι οι τιμές που μπορεί να πάρει μία λογική έκφραση;
2. Ποιοι είναι οι βασικοί λογικοί τελεστές; Αναφέρατε δύο παραδείγματα για τη χρήση του καθενός;
3. Ποια είναι η σύνταξη της εντολής ΑΝ;
4. Ποια είναι η διαφορά της εντολής ΑΝ- ΑΛΛΙΩΣ και της ΑΝ- ΑΛΛΙΩΣ_ΑΝ;
5. Τι είναι τα εμφωλευμένα ΑΝ;
6. Πότε χρησιμοποιείται η εντολή ΕΠΙΛΕΞΕ;

7. Ποιες οι εντολές επανάληψης;
8. Ποιες οι διαφορές της εντολής ΟΣΟ και της εντολής ΜΕΧΡΙΣ_ΟΤΟΥ;
9. Πώς συντάσσεται η εντολή ΓΙΑ;
10. Ποια η βασική διαφορά της εντολής ΓΙΑ από τις άλλες δύο εντολές επανάληψης;

Βιβλιογραφία

1. Ι. Κάβουρας, *Δομημένος προγραμματισμός με Pascal,* Κλειδάριθμος, Αθήνα, 1997.
2. Κ. Γιαλούρης-Κ. Σταθόπουλος, *Προγραμματισμός σε Turbo Pascal,* Αθήνα, 1996.
3. Χρ. Κοίλιας, *Η QuickBasic και οι εφαρμογές της*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.
4. R. Shackelford, *Introduction to Computing and Algorithms*, Addison-Wesley, USA, 1998.
5. S. Leestma-L.Nyhoff, *Turbo Pascal, Programming and Solving*, McMillan, New York, 1990.
6. N. Wirth, *Systematic Programming: An introduction*, Prentice Hall, 1973.

ΚΕΦΑΛΑΙΟ 9ο

ΠΙΝΑΚΕΣ

ΠΕΡΙΕΧΟΜΕΝΑ

- Μονοδιάστατοι πίνακες
- Πότε πρέπει να χρησιμοποιούνται πίνακες
- Πολυδιάστατοι πίνακες
- Τυπικές επεξεργασίες πινάκων

Εισαγωγή

Η χρήση των μεταβλητών με δείκτες στην άλγεβρα είναι ένας ιδιαίτερα δυναμικός τρόπος για τη διαχείριση μεγάλου αριθμού δεδομένων ιδίου τύπου. Οι γλώσσες προγραμματισμού, δανείζονται την έννοια των μεταβλητών με δείκτες και χρησιμοποιούν τους πίνακες για τον ίδιο λόγο.

Στο κεφάλαιο αυτό παρουσιάζονται οι έννοιες του πίνακα καθώς και οι βασικοί τρόποι επεξεργασίας τους από τη **ΓΛΩΣΣΑ**.

Παρουσιάζονται οι μονοδιάστατοι καθώς και οι πολυδιάστατοι πίνακες, ο τρόπος με τον οποίο ορίζονται και χρησιμοποιούνται και τέλος συζητούνται οι πλέον κοινές διαδικασίες πάνω σε πίνακες, η εύρεση μεγίστου και ελαχίστου, η αναζήτηση, η ταξινόμηση και η συγχώνευση πινάκων.

Στόχοι

Να είναι σε θέση ο μαθητής

- Να επιλέγει το είδος του πίνακα.
- Να ορίζει τους πίνακες σε ένα πρόγραμμα.
- Να εισάγει, να επεξεργάζεται και να τυπώνει τα στοιχεία ενός πίνακα.
- Να αποφασίζει αν είναι απαραίτητη η χρήση πίνακα.
- Να αναφέρει τις βασικές επεξεργασίες σε ένα πίνακα.
- Να αναζητά και να ταξινομεί τα στοιχεία ενός πίνακα.

Προερωτήσεις

- ✓ Πώς μπορούν να αποθηκευτούν πολλά παρόμοια δεδομένα στον υπολογιστή, για παράδειγμα τα ονόματα όλων των μαθητών μίας τάξης;
- ✓ Για ποιο λόγο χρησιμοποιούνται στην άλγεβρα οι μεταβλητές με δείκτες;
- ✓ Σε ποιες περιπτώσεις χρειάζεται η ταξινόμηση κάποιων δεδομένων;
- ✓ Είναι εύκολη η αναζήτηση ενός συγκεκριμένου δεδομένου σε μη ταξινομημένα δεδομένα;

9.1. Μονοδιάστατοι πίνακες.

Πολλά από τα προβλήματα τα οποία παρουσιάστηκαν στα προηγούμενα κεφάλαια, απλά επεξεργάζονται μία σειρά δεδομένων.

Διαβάζουν ένα δεδομένο κάθε φορά, το εκχωρούν σε μία μεταβλητή, εκτελούν τους αντίστοιχους υπολογισμούς και στη συνέχεια επαναλαμβάνεται η ίδια διαδικασία μέχρι να τελειώσουν όλα τα δεδομένα.

Για παράδειγμα ένα πρόγραμμα το οποίο διαβάζει τις θερμοκρασίες διαφόρων ημερών του μήνα, έστω 30, και υπολογίζει τη μέση θερμοκρασία, μπορεί πολύ απλά να γραφεί ως εξής

```
….
Σύνολο <- 0
ΓΙΑ Ημέρα ΑΠΟ 1 ΜΕΧΡΙ 30
    ΔΙΑΒΑΣΕ Θερμοκρασία
    Σύνολο <- Σύνολο+Θερμοκρασία
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
Μέση <-  Σύνολο /30
…..
```

Χρησιμοποιώντας λοιπόν μόνο μία μεταβλητή, τη μεταβλητή *Θερμοκρασία*, το πρόβλημα λύνεται πολύ απλά και το αντίστοιχο πρόγραμμα είναι σύντομο και κατανοητό.

Αν όμως στο προηγούμενο πρόγραμμα ζητείται και ο αριθμός των ημερών που η θερμοκρασία ήταν κατώτερη της μέσης, τότε η σύγκριση αυτή πρέπει να γίνει μετά τον υπολογισμό της μέσης θερμοκρασίας. Αυτό σημαίνει ότι όλες οι θερμοκρασίες πρέπει να επαναεισαχθούν για να συγκριθούν με τη μέση.

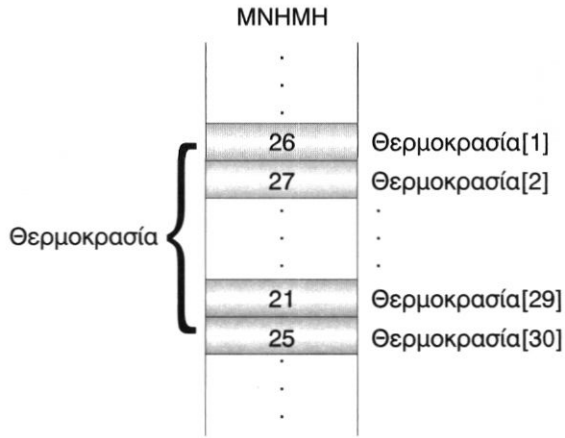
Μία άλλη λύση είναι να καταχωρηθεί κάθε θερμοκρασία σε διαφορετική μεταβλητή, έτσι ώστε κάθε τιμή που εισάγεται να διατηρείται στη μνήμη και να μπορεί να συγκριθεί με τη μέση, αφού αυτή υπολογιστεί. Τότε όμως πρέπει να δημιουργηθούν 30 διαφορετικές μεταβλητές Θερμοκρασία1, Θερμοκρασία2,..., Θερμοκρασία30. Για να γραφεί το πρόγραμμα χρειάζονται τριάντα εντολές ΔΙΑΒΑΣΕ και τριάντα εντολές ΑΝ.

Αν και αυτή η λύση είναι σωστή και πρακτική για μικρό αριθμό δεδομένων, προφανώς δεν εξυπηρετεί την επεξεργασία μεγάλου αριθμού δεδομένων.

Η καλύτερη λύση στο πρόβλημα αυτό είναι η χρήση μεταβλητής με δείκτες, έννοια που είναι γνωστή από τα μαθηματικά και υλοποιείται στον προγραμματισμό με τη δομή δεδομένων του **πίνακα.** Χρησιμοποιείται λοι-

πόν μόνο ένα όνομα *Θερμοκρασία*, που αναφέρεται και στις τριάντα διαφορετικές θερμοκρασίες.

Το όνομα του πίνακα καθορίζει μία ομάδα διαδοχικών θέσεων στη μνήμη. Κάθε συγκεκριμένη θέση μνήμης καλείται στοιχείο του πίνακα και προσδιορίζεται από την τιμή ενός δείκτη, όπως φαίνεται και στο σχήμα 9.1.



Σχ. 9.1. Ο πίνακας Θερμοκρασία

Οι πίνακες που χρησιμοποιούν ένα μόνο δείκτη για την αναφορά των στοιχείων τους, ονομάζονται **μονοδιάστατοι** πίνακες.

Το όνομα του πίνακα μπορεί να είναι οποιοδήποτε δεκτό όνομα της **ΓΛΩΣΣΑΣ** και ο δείκτης είναι μία ακέραια έκφραση, σταθερή ή μεταβλητή που περικλείεται μέσα στα σύμβολα [και]. Το στοιχείο *Θερμοκρασία[2],* εκφράζει τη θερμοκρασία της δεύτερης ημέρας, αναφέρεται στο δεύτερο στοιχείο του πίνακα *Θερμοκρασία* και έχει την τιμή 27.

Ο δείκτης είναι μία μεταβλητή που μπορεί να έχει οποιοδήποτε δεκτό όνομα. Είναι σύνηθες όμως στον προγραμματισμό ως δείκτες να χρησιμοποιούνται οι μεταβλητές i, j, k.

Γενικότερα το στοιχείο Θερμοκρασία[i] αναφέρεται στο i-στό στοιχείο του πίνακα.

Κάθε πίνακας πρέπει υποχρεωτικά να περιέχει δεδομένα του ιδίου τύπου, δηλαδή ακέραια, πραγματικά, λογικά, ή αλφαριθμητικά. Ο τύπος του πίνακα δηλώνεται μαζί με τις άλλες μεταβλητές του προγράμματος στο τμήμα δήλωσης μεταβλητών. Εκτός από τον τύπο του πίνακα πρέπει να δηλώνεται και ο αριθμός των στοιχείων που περιέχει ή καλύτερα ο μεγαλύτερος αριθμός στοιχείων που μπορεί να έχει ο συγκεκριμένος πίνακας και αυτό για να δεσμευτούν οι αντίστοιχες συνεχόμενες θέσεις μνήμης.

Για παράδειγμα

```
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ:Θερμοκρασία [30]
```

Πίνακας είναι ένα σύνολο αντικειμένων ίδιου τύπου, τα οποία αναφέρονται με ένα κοινό όνομα. Κάθε ένα από τα αντικείμενα που απαρτίζουν τον πίνακα λέγεται **στοιχείο** του πίνακα. Η αναφορά σε ατομικά στοιχεία του πίνακα γίνεται με το όνομα του πίνακα ακολουθούμενο από ένα δείκτη.

Παράδειγμα 1

Χρησιμοποιώντας μεταβλητές με δείκτες για το προηγούμενο παράδειγμα έχουμε το εξής πρόγραμμα

```
ΠΡΟΓΡΑΜΜΑ Θερμοκρασίες
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ: Θερμοκρασία[30], Μέση, Σύνολο
    ΑΚΕΡΑΙΕΣ: i, Ημέρες
ΑΡΧΗ
    Σύνολο <- 0
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 30
        ΓΡΑΨΕ  'Δώσε τη θερμοκρασία'
        ΔΙΑΒΑΣΕ  Θερμοκρασία[i]
        Σύνολο <- Σύνολο+ Θερμοκρασία[i]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    Μέση <- Σύνολο/30
    Ημέρες <- 0
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 30
        ΑΝ Θερμοκρασία[i] < Μέση ΤΟΤΕ
            Ημέρες <- Ημέρες+1
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Μέση Θερμοκρασία:', Μέση
    ΓΡΑΨΕ 'Ημέρες με μικρότερη θερμοκρασία', Ημέρες
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Με τη χρήση του πίνακα, όλα τα δεδομένα καταχωρούνται κάτω από το ίδιο όνομα μεταβλητής, στο παράδειγμα Θερμοκρασία. Η ανάγνωση όλων των δεδομένων απλοποιείται, αφού είναι μόνο μία εντολή, **ΔΙΑΒΑΣΕ** Θερμοκρασία[i], η οποία βρίσκεται μέσα σε ένα βρόχο και επαναλαμβάνε-

Η ανάγνωση, η επεξεργασία και η εκτύπωση των στοιχείων των πινάκων γίνεται πάντοτε από βρόχους, οι οποίοι επαναλαμβάνονται προκαθορισμένο αριθμό φορών, όσα είναι τα στοιχεία του πίνακα και υλοποιούνται καλύτερα στον προγραμματισμό με την εντολή επανάληψης ΓΑ.

ται όσες φορές απαιτείται όπως και ο υπολογισμός του αθροίσματος, Σύνολο<-Σύνολο+ Θερμοκρασία[i].

Οι τιμές μετά τον υπολογισμό της μέσης τιμής δεν χάνονται, αφού βρίσκονται στα στοιχεία του πίνακα και ξαναχρησιμοποιούνται για την εύρεση του αριθμού των ημερών, που η θερμοκρασία είναι μικρότερη από τη μέση. Ο υπολογισμός των ημερών είναι μόνο μία εντολή ΑΝ, η οποία βρίσκεται σε ένα βρόχο και επαναλαμβάνεται 30 φορές.

Παράδειγμα 2

Να γραφεί πρόγραμμα το οποίο να υπολογίζει τα βασικά στατιστικά μεγέθη, τη μέση τιμή, την τυπική απόκλιση και τη διάμεσο τιμή Ν ακεραίων αριθμών, όπου το Ν είναι από 2 μέχρι 100.

Τα δεδομένα εισάγονται από το πληκτρολόγιο και καταχωρούνται στον πίνακα Χ.

Η μέση τιμή **μ** δίνεται από τον τύπο $\mu = \frac{\sum_{i=1}^{N} Xi}{N}$

Η τυπική απόκλιση **σ** δίνεται από τον τύπο $\sigma^2 = \frac{\sum_{i=1}^{N} Xi^2}{N} - \mu^2$

Για να βρεθεί η διάμεσος τιμή πρέπει υποχρεωτικά οι αριθμοί να ταξινομηθούν κατά αύξουσα σειρά. Τότε διάμεσος τιμή, είναι η τιμή για την οποία οι μισοί αριθμοί είναι μικρότεροι και οι άλλοι μισοί μεγαλύτεροι. Στην περίπτωση που το πλήθος των αριθμών είναι περιττό, τότε διάμεσος είναι ο μεσαίος, ενώ στην περίπτωση που είναι άρτιο, τότε διάμεσος είναι το ημιάθροισμα των δύο μεσαίων αριθμών.

Η ταξινόμηση των στοιχείων γίνεται με τη μέθοδο ταξινόμησης ευθείας ανταλλαγής, η οποία παρουσιάστηκε στο κεφάλαιο 3.

```
ΠΡΟΓΡΑΜΜΑ Στατιστική

ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ : i, N, X[100], Άθροισμα,
               Άθροισμα_2, Βοηθητική
    ΠΡΑΓΜΑΤΙΚΕΣ: ΜΤ, Τυπ_Απόκλιση, Διάμεσος

ΑΡΧΗ
! Εισαγωγή δεδομένων
ΓΡΑΨΕ  'Δώσε το πλήθος των αριθμών (μέγιστο 100)'
```

```
ΔΙΑΒΑΣΕ  N
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ  N
   ΓΡΑΨΕ 'Δώσε τον ',i,'-το αριθμό'
   ΔΙΑΒΑΣΕ  X[i]
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
! Υπολογισμός αθροισμάτων
Άθροισμα <- 0
Άθροισμα_2 <- 0
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ  N
   ΓΡΑΨΕ 'Δώσε τον', i, '-το αριθμό'
   ΔΙΑΒΑΣΕ X[i]
    Άθροισμα <- Άθροισμα + X[i]
    Άθροισμα_2 <- Άθροισμα_2 + X[i]^2
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ

! Υπολογισμός μέσου όρου
ΜΤ <- Άθροισμα/N
!Υπολογισμός τυπικής απόκλισης
Τυπ_Απόκλιση <- T_P(Άθροισμα_2/N - ΜΤ^2)
!Ταξινόμηση των στοιχείων του πίνακα
ΓΙΑ i ΑΠΟ 2 ΜΕΧΡΙ N
    ΓΙΑ j ΑΠΟ N ΜΕΧΡΙ i ΜΕ_ΒΗΜΑ -1
    ΑΝ X[j-1] > X[j] ΤΟΤΕ
        ! Αντιμετάθεση των στοιχείων j και j-1
        Βοηθητική <- X[j-1]
        X[j-1] <- X[j]
        X[j] <- Βοηθητική
    ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ !  j
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ !  i
!Υπολογισμός διαμέσου
    ΑΝ N MOD 2 =0 ΤΟΤΕ
        Διάμεσος <- (X[N/2]+X[N/2+1])/2
    ΑΛΛΙΩΣ
        Διάμεσος <- X[(N+1)/2]
    ΤΕΛΟΣ_ΑΝ
! Εκτύπωση αποτελεσμάτων
ΓΡΑΨΕ 'ΑΠΟΤΕΛΕΣΜΑΤΑ'
ΓΡΑΨΕ '============'
ΓΡΑΨΕ 'Πλήθος τιμών =', N
ΓΡΑΨΕ 'Μέση τιμή = ', ΜΤ
ΓΡΑΨΕ 'Τυπική απόκλιση = ', Τυπ_Απόκλιση
ΓΡΑΨΕ 'Διάμεσος = ', Διάμεσος
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Στο πρόγραμμα αυτό δεν γίνεται έλεγχος για μηδενική τιμή του N

Προγραμματιστικό περιβάλλον Basic

```
' Στατιστική
DEFINT A-Z
DIM x(100)
CLS
sum = 0: sum2 = 0
' Εισαγωγή δεδομένων
INPUT "N=", n
FOR i = 1 TO n
  PRINT "Δώσε τον "; i; "-το αριθμό : ";
  INPUT "", x(i)
NEXT i
' Υπολογισμός αθροισμάτων
FOR i = 1 TO n
  sum = sum + x(i)
  sum2 = sum2 + x(i) ^ 2
NEXT i
mt! = sum / n     ' Μέση τιμή
ta! = SQR(sum2 / n - mt! ^ 2) 'Τυπ.Απόκλ.
' Ταξινόμηση πίνακα
FOR i = 2 TO n
  FOR j = n TO i STEP -1
    IF x(j - 1) > x(j) THEN SWAP x(j - 1), x(j)
  NEXT j
NEXT i
' Υπολογισμός διαμέσου
IF n MOD 2 = 0 THEN
   median = (x(n / 2) + x(n / 2 + 1)) / 2
ELSE
   median = x((n + 1) / 2)
END IF
PRINT "     ΑΠΟΤΕΛΕΣΜΑΤΑ"
PRINT "======================"
PRINT "Πλήθος τιμών     ="; n
PRINT "Μέση τιμή        ="; mt!
PRINT "Τυπική απόκλιση ="; ta!
PRINT "Διάμεσος         ="; median
END
```

9.2. Πότε πρέπει να χρησιμοποιούνται πίνακες

Η χρήση πινάκων είναι ένας βολικός τρόπος για τη διαχείριση πολλών δεδομένων ιδίου τύπου, αλλά συχνά η χρήση τους είναι περιττή και επιζήμια στην ανάπτυξη του προγράμματος.

Πέρα από τα πλεονεκτήματα που αναφέρθηκαν, υπάρχουν και δύο μειονεκτήματα από τη χρήση πινάκων.

Οι πίνακες απαιτούν μνήμη. Κάθε πίνακας δεσμεύει από την αρχή του προγράμματος πολλές θέσεις μνήμης. Σε ένα μεγάλο και σύνθετο πρόγραμμα η άσκοπη χρήση μεγάλων πινάκων μπορεί να οδηγήσει ακόμη και σε αδυναμία εκτέλεσης του προγράμματος.

Οι πίνακες περιορίζουν τις δυνατότητες του προγράμματος. Στο προηγούμενο πρόγραμμα του υπολογισμού των στατιστικών μεγεθών, υπάρχει ανώτατο όριο στο πλήθος των αριθμών ίσο με 100. Αυτό γιατί οι πίνακες είναι στατικές δομές και το μέγεθος τους πρέπει να δηλώνεται στην αρχή του προγράμματος, ενώ παραμένει υποχρεωτικά σταθερό κατά την εκτέλεση του προγράμματος.

Η απόφαση για την χρήση ή όχι πίνακα για την διαχείριση των δεδομένων είναι κυρίως θέμα εμπειρίας στον προγραμματισμό.

Γενικά, αν τα δεδομένα που εισάγονται σε ένα πρόγραμμα πρέπει να διατηρούνται στη μνήμη μέχρι το τέλος της εκτέλεσης, τότε η χρήση πινάκων βοηθάει ή συχνά είναι απαραίτητη για την επίλυση του προβλήματος. Σε άλλη περίπτωση μπορεί να αποφεύγεται η χρήση τους.

Ας επιστρέψουμε στο προηγούμενο παράδειγμα.

Για τον υπολογισμό της μέσης τιμής και της τυπικής απόκλισης δεν είναι απαραίτητο να διατηρούνται οι τιμές στη μνήμη. Το πρόγραμμα θα δούλευε το ίδιο καλά και χωρίς τη χρήση πινάκων, αλλά με τη χρήση μίας και μόνο μεταβλητής. Ο υπολογισμός όμως της διάμεσης τιμής, που προϋποθέτει την ταξινόμηση των δεδομένων, απαιτεί τη χρήση πίνακα. Αν λοιπόν το πρόβλημα απαιτούσε μόνο τον υπολογισμό του μέσου όρου και της τυπικής απόκλισης, θα ήταν προτιμότερη μία λύση χωρίς τη χρήση πινάκων.

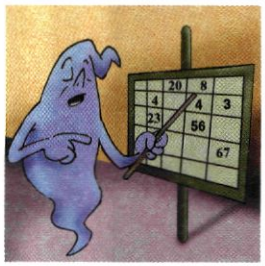
9.3. Πολυδιάστατοι πίνακες

Στο προηγούμενο παράδειγμα υπήρχαν 30 τιμές της θερμοκρασίας, μία για κάθε ημέρα του μήνα και για την επίλυση του χρησιμοποιήθηκε ένας πί-

νακας 30 θέσεων, ο πίνακας *Θερμοκρασία*. Ο πίνακας αυτός περιέχει τις τιμές της θερμοκρασίας για κάθε ημέρα μίας πόλης και ο δείκτης δείχνει την ημέρα.

Έστω ότι οι θερμοκρασίες δίνονται από τον παρακάτω πίνακα.

ΗΜΕΡΑ	ΠΟΛΗ			
	1	2	...	10
1	25	21	...	32
2	26	22	...	31
....	..	..	..	..
30	27	23	...	30



Ο πίνακας αυτός έχει τις θερμοκρασίες για 30 ημέρες αλλά για δέκα διαφορετικές πόλεις, δηλαδή υπάρχουν συνολικά 300 τιμές θερμοκρασίας. Για να καθοριστεί κάθε στοιχείο δεν αρκεί μόνο ένας δείκτης, αλλά απαιτούνται δύο δείκτες, ο ένας για την ημέρα και ο δεύτερος για την πόλη. Για παράδειγμα η πρώτη πόλη την τριακοστή ημέρα είχε θερμοκρασία 27.

Για την επεξεργασία των θερμοκρασιών μπορεί να χρησιμοποιηθεί ένας *δισδιάστατος* πίνακας, στον οποίο ο πρώτος δείκτης δείχνει τη γραμμή (στο παράδειγμα την ημέρα) και ο δεύτερος τη στήλη (την πόλη). Το στοιχείο *Θερμοκρασία[30,1]* έχει την τιμή 27.

Στην γενική περίπτωση κάθε στοιχείο του πίνακα είναι το στοιχείο Θερμοκρασία[i,j] και αναφέρεται στη θερμοκρασία στην i γραμμή και την j στήλη, όπου το i παίρνει τιμή από 1 έως 30 και το j από 1 έως 10. Ο πίνακας *Θερμοκρασία* είναι ένας δισδιάστατος πίνακας 30Χ10.

Παράδειγμα 3

Να γραφεί πρόγραμμα που να υπολογίζει τη μέση θερμοκρασία κάθε πόλης για τον προηγούμενο πίνακα θερμοκρασιών (δίδονται 30 θερμοκρασίες 10 πόλεων). Επίσης, για κάθε πόλη, να υπολογίζει πόσες ημέρες η θερμοκρασία ήταν κατώτερη από την αντίστοιχη μέση.

```
ΠΡΟΓΡΑΜΜΑ Θερμοκρασίες_2
ΜΕΤΑΒΛΗΤΕΣ
    ΠΡΑΓΜΑΤΙΚΕΣ: Θερμοκρασία[30,10], Μέση[10]
    ΑΚΕΡΑΙΕΣ: i, j,  Ημέρες, Σύνολο
ΑΡΧΗ
! Εισαγωγή δεδομένων
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 30
    ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 10
        ΓΡΑΨΕ 'Δώσε τη θερμοκρασία', i, j
        ΔΙΑΒΑΣΕ Θερμοκρασία[i,j]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
!Υπολογισμοί
ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 10
    Σύνολο <- 0
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 30
        Σύνολο <- Σύνολο + Θερμοκρασία[i,j]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    Μέση[j] <- Σύνολο/30
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 10
    Ημέρες <- 0
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 30
        ΑΝ Θερμοκρασία[i,j] < Μέση[j] ΤΟΤΕ
            Ημέρες <- Ημέρες+1
        ΤΕΛΟΣ_ΑΝ
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Μέση Θερμοκρασία ', i, 'Πόλης:', Μέση[j]
    ΓΡΑΨΕ 'Ημέρες με μικρότερη θερμοκρασία', Ημέρες
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Η ανάγνωση, η επεξεργασία καθώς και η εκτύπωση των στοιχείων πολυδιάστατων πινάκων γίνεται πάντοτε από βρόχους, οι οποίοι υλοποιούνται στον προγραμματισμό με εμφωλευμένες εντολές επανάληψης ΓΙΑ.

Προγραμματιστικό περιβάλλον Pascal

```
PROGRAM poleis;
VAR
    temperature: array[1..30,1..10] OF REAL;
    ave:array [1..10] OF REAL;
    total:REAL;
    i,j,days:INTEGER;
```

```
BEGIN
    FOR i:=1 TO 30 DO
        FOR j:=1 TO 10 DO
        BEGIN
            write ('Δώσε τη θερμοκρασία :',i,j,'  ');
            readln (temperature[i,j])
        END;
    FOR j:=1  TO 10  DO
    BEGIN
        total:=0;
        FOR i:=1 TO 30 DO
            total:=total+temperature[i,j];
        ave[j]:=total/30
    END;
    FOR j:=1 TO 10 DO
    BEGIN
        days:=0;
        FOR i:=1 TO 30 DO
            IF temperature[i,j] < ave[j] then
                days:=days+1;
        writeln ('Μέση θερμοκρασία ',i,' πόλης
                                    ',ave[i]:4:1);
        writeln  ('Ημέρες με μικρότερη θερμοκρασία :',
                                                days);
    END
END.
```

Εκτός από μονοδιάστατους και δισδιάστατους πίνακες υπάρχουν πίνακες με περισσότερες διαστάσεις τρισδιάστατοι, τετραδιάστατοι και γενικά πολυδιάστατοι, ανάλογα με τον αριθμό των δεικτών που χρησιμοποιούνται για τον καθορισμό των στοιχείων. Ωστόσο τα περισσότερα προβλήματα αντιμετωπίζονται με τη χρήση πινάκων μονοδιάστατων ή δισδιάστατων.

Στο προηγούμενο παράδειγμα έστω ότι έχουμε θερμοκρασίες για κάθε μία πόλη, για κάθε ημέρα αλλά και για κάθε έτος. Τότε κάθε στοιχείο χρειάζεται τρεις δείκτες. Έναν που θα δείχνει το έτος, έναν που θα δείχνει την πόλη και έναν που θα δείχνει την ημέρα. Για παράδειγμα η θερμοκρασία της πρώτης ημέρας, της πρώτης πόλης και του πρώτου έτους είναι 25 και είναι το στοιχείο *Θερμοκρασία [1,1,1]*.

2001	ΠΟΛΗ			
ΗΜΕΡΑ	1	2	...	10
1	25	21	...	32
				31
				...
				30

2000	ΠΟΛΗ			
ΗΜΕΡΑ	1	2	...	10
1	25	21	...	32
				31
				...
				30

1999	ΠΟΛΗ			
ΗΜΕΡΑ	1	2	...	10
1	25	21	...	32
2	26	22	...	31
...	...	...	...	...
30	27	23	...	30

Παράδειγμα 4.

Να γραφεί πρόγραμμα το οποίο:

α. Να διαβάζει τα ονόματα δέκα κινηματογράφων και τις αντίστοιχες εισπράξεις τους για κάθε ημέρα μίας εβδομάδας.

β. Να υπολογίζει και να εκτυπώνει το άθροισμα των εισπράξεων κάθε κινηματογράφου, καθώς και τον κινηματογράφο με τη μέγιστη συνολική είσπραξη.

γ. Να υπολογίζει και να εκτυπώνει το άθροισμα των εισπράξεων κάθε ημέρας, καθώς και την ημέρα με τη μέγιστη συνολική είσπραξη.

Για την επίλυση του προβλήματος πρέπει να χρησιμοποιηθούν δύο πίνακες. Ο πρώτος θα περιέχει μόνο τα ονόματα των κινηματογράφων, θα είναι δηλαδή ένας μονοδιάστατος πίνακας χαρακτήρων με δέκα γραμμές. Ο δεύτερος θα περιέχει τις εισπράξεις, θα είναι ένας πίνακας δισδιάστατος ακεραίων αριθμών με δέκα γραμμές, μία για κάθε κινηματογράφο και επτά στήλες, μία για κάθε ημέρα.

Το πρόγραμμα ουσιαστικά αποτελείται από τρία τμήματα:

1. Την ανάγνωση των δεδομένων και την καταχώρηση του στους αντίστοιχους πίνακες, *Ονόματα* και *Εισπράξεις*.
2. Τον υπολογισμό του συνόλου των εισπράξεων ανά κινηματογράφο και την εύρεση της μέγιστης συνολικής είσπραξης, δηλαδή του αθροίσματος των γραμμών του πίνακα.
3. Τον υπολογισμό του συνόλου των εισπράξεων ανά ημέρα και την εύρεση της μέγιστης συνολικής είσπραξης, δηλαδή του αθροίσματος των στηλών του πίνακα.

Προσέξτε τις διαφορές του δεύτερου και του τρίτου τμήματος. Και οι δύο χρησιμοποιούν δύο εμφωλευμένους βρόχους, ένα για τις γραμμές και ένα για τις στήλες αλλά σε διαφορετική σειρά.

```
ΠΡΟΓΡΑΜΜΑ Κινηματογράφοι
ΜΕΤΑΒΛΗΤΕΣ
    ΑΚΕΡΑΙΕΣ: Εισπράξεις [10,7], i,j, Άθροισμα, Μέγιστο,
                                                    & Θέση
    ΧΑΡΑΚΤΗΡΕΣ: Ονόματα[10]
ΑΡΧΗ
! Εισαγωγή δεδομένων και εκχώρηση τους σε δύο πίνακες
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    ΓΡΑΨΕ 'Δώσε το όνομα του ',i,'  κινηματογράφου'
    ΔΙΑΒΑΣΕ Ονόματα[i]
    ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 7
        ΓΡΑΨΕ 'Δώσε την ', j,'-η είσπραξη'
        ΔΙΑΒΑΣΕ Εισπράξεις[i,j]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
!Άθροισμα των στοιχείων του πίνακα Εισπράξεις ανά γραμμή
!και υπολογισμός του μέγιστου αθροίσματος
Μέγιστο <- 0
ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
    Άθροισμα <- 0
    ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 7
        Άθροισμα <- Άθροισμα+Εισπράξεις[i,j]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Συνολ. είσπραξη ',Ονόματα[i], '=', Άθροισμα
    ΑΝ Μέγιστο < Άθροισμα ΤΟΤΕ
        Μέγιστο <- Άθροισμα
        Θέση <- i
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
```

```
ΓΡΑΨΕ 'Μέγ.συνολ.είσπραξη ',Μέγιστο,'στον ',Ονόματα[Θέση]
!Άθροισμα των στοιχείων του πίνακα Εισπράξεις ανά στήλη
!και υπολογισμός του μέγιστου αθροίσματος
Μέγιστο <- 0
ΓΙΑ j ΑΠΟ 1 ΜΕΧΡΙ 7
    Άθροισμα <- 0
    ΓΙΑ i ΑΠΟ 1 ΜΕΧΡΙ 10
        Άθροισμα <- Άθροισμα + Εισπράξεις[i,j]
    ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
    ΓΡΑΨΕ 'Συνολ. είσπραξη',j,'-ης ημέρας =', Άθροισμα
    ΑΝ Μέγιστο < Άθροισμα ΤΟΤΕ
        Μέγιστο <- Άθροισμα
        Θέση <- j
    ΤΕΛΟΣ_ΑΝ
ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
ΓΡΑΨΕ 'Μέγ.συν.είσπραξη ',Μέγιστο,'την ',Θέση,'-η ημέρα'
ΤΕΛΟΣ_ΠΡΟΓΡΑΜΜΑΤΟΣ
```

Προγραμματιστικό περιβάλλον Pascal

```
PROGRAM cinemas;
VAR
    i,j,k,max,sum:INTEGER;
    tickets:ARRAY[1..10,1..7] OF INTEGER;
    name:ARRAY [1..10] OF STRING;

BEGIN
    FOR i:=1 TO 10 do
    BEGIN
        write ('Δώσε το όνομα : ');
        readln (name[i]);
        FOR j:=1 TO 7 DO
            BEGIN
                write ('Δώσε την ', j,' η είσπραξη : ');
                readln(tickets[i,j]);
            END;
    END;
    max:=0;
    FOR i:=1 TO 10 DO
    BEGIN
        sum:=0;
        FOR j:=1 TO 7 DO
            sum:=sum+tickets[i,j];
```

```
        writeln('Σύνολο ', name[i],' = ', sum);
        IF max<sum THEN
        BEGIN
            max:=sum;
            k:=i;
        END;
    END;
    writeln ('Μέγιστο ', max, ' στον  ', name[k]);

    max:=0;
    FOR j:=1 TO 7 DO
    BEGIN
        sum:=0;
        FOR i:=1 TO 10 DO
            sum:=sum+tickets[i,j];
        writeln('Σύνολο ',j,' ημέρας = ', sum);
        IF max<sum THEN
        BEGIN
            max:=sum;
            k:=j;
        END;
    END;
    writeln ('Μέγιστο ', max, ' την ', k);
END.
```

9.4. Τυπικές επεξεργασίες πινάκων

Τα προγράμματα τα οποία χρησιμοποιούν πίνακες πολύ συχνά απαιτούν συγκεκριμένες επεξεργασίες στα στοιχεία του πίνακα. Οι τυπικές αυτές επεξεργασίες είναι:

- Υπολογισμός αθροισμάτων στοιχείων του πίνακα.
- Εύρεση του μέγιστου ή του ελάχιστου στοιχείου.
- Ταξινόμηση των στοιχείων του πίνακα.
- Αναζήτηση ενός στοιχείου του πίνακα.
- Συγχώνευση δύο πινάκων.

Μερικές από αυτές τις επεξεργασίες παρουσιάστηκαν ήδη στα παραδείγματα αυτού του κεφαλαίου. Για τις διαδικασίες της ταξινόμησης και της αναζήτησης έχουν παρουσιαστεί μερικοί αλγόριθμοι, οι οποίοι αναπτύχθη-

καν στο αντίστοιχο κεφάλαιο των αλγορίθμων, όπου συζητήθηκαν τα πλεονεκτήματα και τα μειονεκτήματα τους.

Συνοπτικά έχουμε:

Υπολογισμός αθροισμάτων στοιχείων του πίνακα.

Πολύ συχνά απαιτείται ο υπολογισμός του αθροίσματος στοιχείων του πίνακα που έχουν κοινά χαρακτηριστικά για παράδειγμα βρίσκονται στην ίδια στήλη ή στην ίδια γραμμή. Τα παραδείγματα που παρουσιάστηκαν σε αυτό το κεφάλαιο απαιτούσαν τον υπολογισμό των αθροισμάτων των στοιχείων του πίνακα τόσο ανά γραμμή όσο και ανά στήλη.

Εύρεση του μέγιστου ή του ελάχιστου στοιχείου.

Η εύρεση του μέγιστου ή του ελάχιστου στοιχείου ενός πίνακα παρουσιάστηκε στα προηγούμενα παραδείγματα. Αν ο πίνακας δεν είναι ταξινομημένος, τότε πρέπει να συγκριθούν τα στοιχεία ένα προς ένα, για να βρεθεί το μέγιστο ή το ελάχιστο. Αν ο πίνακας είναι ταξινομημένος, τότε προφανώς το μέγιστο και το ελάχιστο βρίσκονται στα δύο ακριανά στοιχεία του πίνακα.

Ταξινόμηση των στοιχείων του πίνακα.

Στο κεφάλαιο 3 αναφέρθηκε η μέθοδος ταξινόμησης της ευθείας ανταλλαγής, η οποία χρησιμοποιήθηκε και στο παράδειγμα 2.

Η μέθοδος αυτή είναι από τις απλούστερες αλλά δεν είναι η πιο αποδοτική. Υπάρχουν πολλές άλλες μέθοδοι ταξινόμησης καθώς και παραλλαγές αυτών.

Η επιλογή του καλύτερου αλγόριθμου εξαρτάται κυρίως από το πλήθος των στοιχείων του πίνακα και την αρχική τους διάταξη, αν δηλαδή ο πίνακας είναι τελείως αταξινόμητος ή μερικώς ταξινομημένος.

Αναζήτηση ενός στοιχείου του πίνακα.

Δύο είναι οι πλέον διαδεδομένοι αλγόριθμοι αναζήτησης:

- ✓ Η σειριακή αναζήτηση
- ✓ Η δυαδική αναζήτηση

Η σειριακή μέθοδος αναζήτησης είναι η πιο απλή, αλλά και η λιγότερη αποτελεσματική μέθοδος. Χρησιμοποιείται όμως υποχρεωτικά για πίνακες που δεν είναι ταξινομημένοι. Αντίθετα η δυαδική αναζήτηση χρησιμοποιείται μόνο σε ταξινομημένους πίνακες και είναι σαφώς αποδοτικότερη από τη σειριακή μέθοδο.

Συγχώνευση δύο πινάκων.

Η συγχώνευση είναι μία από τις βασικές λειτουργίες σε πίνακες.

Σκοπός της είναι η δημιουργία από τα στοιχεία δύο (ή περισσότερων) ταξινομημένων πινάκων ενός άλλου, που είναι και αυτός ταξινομημένος.

Ανακεφαλαίωση

Στο κεφάλαιο αυτό παρουσιάστηκαν οι πίνακες και πώς η ΓΛΩΣΣΑ χειρίζεται τους πίνακες. Ο πίνακας είναι μία ομάδα μεταβλητών ιδίου τύπου που αναφέρονται με ένα κοινό όνομα και αποθηκεύονται σε διαδοχικές θέσεις στη μνήμη. Για την πρόσβαση σε ένα ατομικό στοιχείο του πίνακα πρέπει να γραφεί το όνομα του πίνακα ακολουθούμενο από ένα δείκτη (μεταβλητή ή σταθερά). Οι πίνακες μπορεί να είναι μονοδιάστατοι, δισδιάστατοι ή γενικότερα πολυδιάστατοι. Ο αριθμός των δεικτών καθορίζει τη διάσταση του πίνακα. Η επεξεργασία πινάκων γίνεται συνήθως με τη χρήση εντολών ΓΙΑ.

Οι τυπικές επεξεργασίες που γίνονται σε πίνακες περιλαμβάνουν την αναζήτηση, την ταξινόμηση και τη συγχώνευση πινάκων. Για αυτές τις επεξεργασίες έχουν αναπτυχθεί αρκετοί αλγόριθμοι και η μελέτη τους αποτελεί έναν από τους σημαντικούς τομείς της αλγοριθμικής.

Λέξεις κλειδιά

Μεταβλητή με δείκτη, Πίνακας, Στοιχείο πίνακα, Αναζήτηση, Ταξινόμηση, Συγχώνευση

Ερωτήσεις - Θέματα για συζήτηση

1. Τι ονομάζεται πίνακας;
2. Για ποιο λόγο χρησιμοποιούνται οι πίνακες;
3. Τι μπορεί να είναι οι δείκτες των πινάκων;
4. Ποια η διαφορά του πίνακα και του στοιχείου ενός πίνακα;
5. Πού ορίζεται η διάσταση του πίνακα;
6. Τι είδους δομή είναι ο πίνακας;
7. Ποιοι πίνακες ονομάζονται μονοδιάστατοι;
8. Δώσε ένα παράδειγμα ενός τρισδιάστατου πίνακα.

9. Πού αποθηκεύονται τα στοιχεία ενός πίνακα;
10. Ποια τα μειονεκτήματα της χρήσης πινάκων;
11. Ποιες οι τυπικές επεξεργασίες πίνακα;
12. Ποιοι είναι οι πιο διαδεδομένοι αλγόριθμοι αναζήτησης; Ποιες οι διαφορές τους;

Βιβλιογραφία

1. Κ. Αντωνακόπουλος, *Turbo Pascal 6.0 Θεωρία και Πράξη*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1993.
2. *QuickBASIC: Η εργαλειοθήκη του προγραμματιστή*, Κλειδάριθμος, Αθήνα, 1991.
3. Χρ. Κοίλιας, *Η QuickBasic και οι εφαρμογές της*, Εκδόσεις Νέων Τεχνολογιών, Αθήνα, 1992.
4. R. Shackelford, *Introduction to Computing and Algorithms*, Addison-Wesley, USA, 1998.
5. S. Leestma-L.Nyhoff, *Turbo Pascal, Programming and Solving*, McMillan, New York, 1990.
6. N. Wirth, *Systematic Programming: An introduction*, Prentice Hall, 1973.

Διευθύνσεις Διαδικτύου

- http://qbasic.com/

Περιέχει εκπαιδευτικό οδηγό, κώδικα πολλών ασκήσεων και γενικές πληροφορίες για την Qbasic.

Επίσης στο διαδίκτυο παρουσιάζουν ενδιαφέρον ομάδες νέων (Usenet), σχετικές με τις γλώσσες προγραμματισμού.